BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PATENT ABSTRACTS OF JAPAN

(11)Publication number : **2000-112585**
(43)Date of publication of application : **21.04.2000**

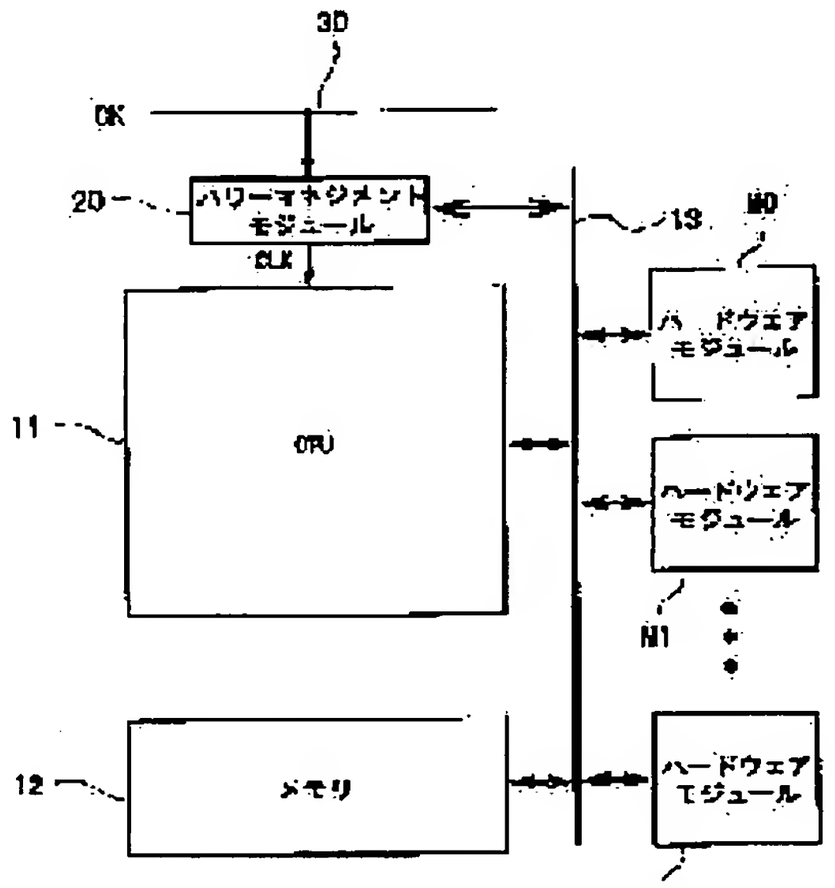
(51)Int.CI. **G06F 1/32**
G06F 1/26
G06F 1/04

(21)Application number : **10-281610**
(22)Date of filing : **02.10.1998**

(71)Applicant : **TOSHIBA CORP**
(72)Inventor : **KAWABE NAOYUKI**
USAMI MASAYOSHI

(54) SYSTEM LSI AND POWER MANAGEMENT METHOD

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a system LSI preventing the deterioration of performance due to the execution of the useless sleep of a CPU, and sharply improving a low power consumption effect.
SOLUTION: This system LSI is provided with one or plural hardware modules M1-Mm for executing prescribed processing and a CPU 11 for controlling the operation of the hardware modules according to a program. This system LSI is also provided with a power management device 20 for controlling the stop and restar of clock supply to the CPU 11. That is, the CPU 11 is provided with a means for executing a sleep instruction for operating the stop of the clock supply to the CPU 11. The power management device 20 stops the clock supply to the CPU 11 based on the execution of the sleep instruction by the CPU 11, and restarts the clock supply to the CPU 11 at the time of receiving the end of the processing of the hardware modules.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of

rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

* NOTICES *

CLAIMS

[Claim(s)]

[Claim 1] The system LSI characterized by forming the power management equipment which controls a halt and resumption of clock supply to Above CPU in the system LSI equipped with one or more hardware modules which perform predetermined processing, and CPU which controls operation of the aforementioned hardware module according to a program.

[Claim 2] It is the system LSI according to claim 1 which prepares a means to execute the sleep instruction for stopping clock supply to Above CPU in Above CPU, and is characterized by making the aforementioned power management equipment the composition which stops the clock supply to Above CPU based on execution of the aforementioned sleep instruction by Above CPU.

[Claim 3] The aforementioned power management equipment is a system LSI according to claim 2 characterized by making it the composition which resumes the clock supply to Above CPU in response to the end of processing of the aforementioned hardware module.

[Claim 4] The aforementioned sleep instruction is a system LSI according to claim 3 which has the 1st field which specifies by the end of processing of which hardware module the clock supply to Above CPU is resumed, and is characterized by making the aforementioned power management equipment the composition which resumes the clock supply to Above CPU with reference to the content of the 1st field of the above.

[Claim 5] The aforementioned sleep instruction is a system LSI according to claim 4 which has the 2nd field which specifies the register which stored the value of the 1st field of the above, and is characterized by making the aforementioned power management equipment the composition which resumes the clock supply to Above CPU with reference to the content of the above 1st and the 2nd field.

[Claim 6] The system LSI according to claim 2 to 5 characterized by making the routine in the aforementioned program used in order to wait for the end of processing of the aforementioned hardware module the composition automatically transposed to the aforementioned sleep instruction.

[Claim 7] It is the system LSI according to claim 1 to 6 characterized by making it the composition which it has the memory connected to Above CPU, and the aforementioned power management equipment stops clock supply in the aforementioned memory at the time of a halt of the clock supply to Above CPU, and resumes clock supply in the aforementioned memory at the time of resumption of the clock supply to Above CPU.

[Claim 8] The aforementioned memory is a system LSI according to claim 7 characterized by forming on the same chip as Above CPU, the aforementioned hardware module, and the aforementioned power management equipment.

[Claim 9] To the sleep instruction which it performs [instruction] in CPU and stops supply of the clock to this CPU When the content which directs the conditions of the operating state of the hardware module for making supply of the clock to CPU resume is included and CPU executes the aforementioned sleep instruction After storing the aforementioned conditions in the register, when the signal which stops supply of the clock to CPU is generated and the operating state of a hardware module agrees with the aforementioned conditions stored in the aforementioned register The power management method

characterized by generating the signal which makes supply of the clock to CPU resume.

[Translation done.]

Translation of JP 2000-112585

* NOTICES *

1. This document has been translated by computer.So the translation may not reflect the original precisely.
2. ** shows the word which can not be translated.**
3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]
[Claim 1] The system LSI characterized by forming the power management equipment which controls a halt and resumption of clock supply to Above CPU in the system LSI equipped with one or more hardware modules which perform predetermined processing, and CPU which controls operation of the aforementioned hardware module according to a program.
[Claim 2] It is the system LSI according to claim 1 which prepares a means to execute the sleep instruction for stopping clock supply to Above CPU in Above CPU, and is characterized by making the aforementioned power management equipment the composition which stops the clock supply to Above CPU based on execution of the aforementioned sleep instruction by Above CPU.
[Claim 3] The aforementioned power management equipment is a system LSI according to claim 2 characterized by making it the composition which resumes the clock supply to Above CPU in response to the end of processing of the aforementioned hardware module.
[Claim 4] The aforementioned sleep instruction is a system LSI according to claim 3 which has the 1st field which specifies by the end of processing of which hardware module the clock supply to Above CPU is resumed, and is characterized by making the aforementioned power management equipment the composition which resumes the clock supply to Above CPU with reference to the contents of the 1st field of the above.
[Claim 5] The aforementioned sleep instruction is a system LSI according to claim 4 which has the 2nd field which specifies the register which stored the value of the 1st field of the above, and is characterized by making the aforementioned power management equipment the composition which resumes the clock supply to Above CPU with reference to the contents of the above 1st and the 2nd field.
[Claim 6] The system LSI according to claim 2 to 5 characterized by making the routine in the aforementioned program used in order to wait for the end of processing of the aforementioned hardware module the composition automatically transposed to the

aforementioned sleep instruction.
[Claim 7] It is the system LSI according to claim 1 to 6 characterized by making it the composition which it has the memory connected to Above CPU, and the aforementioned power management equipment stops clock supply in the aforementioned memory at the time of a halt of the clock supply to Above CPU, and resumes clock supply in the aforementioned memory at the time of resumption of the clock supply to Above CPU.
[Claim 8] The aforementioned memory is a system LSI according to claim 7 characterized by forming on the same chip as Above CPU, the aforementioned hardware module, and the aforementioned power management equipment.
[Claim 9] To the sleep instruction which it performs [instruction] in CPU and stops supply of the clock to this CPU When the contents which direct the conditions of the operating state of the hardware module for making supply of the clock to CPU resume are included and CPU executes the aforementioned sleep instruction After storing the aforementioned conditions in the register, when the signal which stops supply of the clock to CPU is generated and the operating state of a hardware module agrees with the aforementioned conditions stored in the aforementioned register The power management method characterized by generating the signal which makes supply of the clock to CPU resume.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[The technical field to which invention belongs] this invention relates to the system LSI which carried CPU and one or more hardware modules on the chip, and the power management method of realizing the low-power-ization.
[0002]
[Description of the Prior Art] Generally, the conventional system LSI has the composition of having carried on the chip CPU111, two or more hardware modules M0, M1, --, Mn, and the memory 112 that stores an instruction and data, as shown in drawing 20 . CPU111 issues the processing instruction other than the processing which itself performs to hardware modules M0-Mn. The operating state of hardware modules M0-Mn, i.e., execution of the processing which was performing given processing or was given, is completed, and the operating state whether the following processing instruction is waiting or to be among sleep is supervised by CPU111.
[0003] About low-power-izing of this system LSI, the following technology is proposed, for example. (1) CPU111 stops supply of the clock to the hardware modules M0-Mn, and attains low-power-ization in the period when some hardware modules M0-Mn do not need to be operating by changing these hardware modules M0-Mn into a sleep state. (2) If a hardware module M0 - Mn itself end the given processing, low-power-ization will be attained by changing into a sleep state automatically.
[0004]
[Problem(s) to be Solved by the Invention] However, about low-power-izing of the conventional system LSI, the power consumed by CPU occupies many of power consumed with the whole chip, and has been a technical problem with important low-

power-izing of CPU.
[0005] From such a viewpoint, the following can be considered as the technique of changing CPU111 into a sleep state. There is a period when CPU111 cannot perform the next processing until hardware modules M0-Mn end the processing to which predetermined was given fr m relations, such as data dependence, like, when CPU111 performs the next processing as a period which can change CPU111 into a sleep state using the result of the processing which hardware modules M0-Mn performed, for example. Although CPU111 can be changed into a sleep state between the latency times to the processing end of such hardware modules M0-Mn, the method poses a problem.
[0006] In the case where CPU111 is changed into a sleep state with hardware control, there is a possibility of causing the degradation by execution of useless sleep and the fall of the power consumption curtailment effect. It is because sleep may be carried out although it must be able to carry out the Wake rise immediately after changing CPU111 into a sleep state or CPU111 can be changed into a sleep state, since CPU111 cannot predict operation which should be performed to a degree. Moreover, when it is going to control by hardware strictly the timing which carries out sleep, there is also an overhead of area and power consumption or a problem of becoming large, by the hardware.
[0007] On the other hand, the period which changes CPU111 into a sleep state for CPU111 sleep and when carrying out the Wake rise is beforehand estimated by software control, and the Wake rise is carried out, after pointing to the period by sleep instruction and changing into a sleep state. However, when many sleep periods are estimated, sleep will be carried out to a useless period and degradation will be caused after all. On the contrary, when it estimates few, a clock will be supplied in the period which may be carrying out sleep in practice, and there is a problem that the power consumption curtailment effect falls.
[0008] Made in order that this invention might solve the conventional trouble like ****, the purpose is offering the system LSI and the power management method of raising a low-power-ized effect sharply, avoiding the degradation by execution of sleep with useless CPU. Moreover, the other purposes are offering the system LSI which can attain the increase in efficiency of the low-power design in a system LSI.
[0009]
[Means for Solving the Problem] In order to attain the above-mentioned purpose, the feature of the system LSI which is invention according to claim 1 is in the system LSI equipped with one or more hardware modules which perform predetermined processing, and CPU which controls operation of the aforementioned hardware module according to a program to have formed the power management equipment which controls a halt and resumption of clock supply to Above CPU.
[0010] In invention of the claim 1 above-mentioned publication, the feature of the system LSI which is invention according to claim 2 prepares a means execute the sleep instruction for stopping clock supply to Above CPU in Above CPU, and the aforementioned power management equipment is in it for having carried out to the composition which stops the clock supply to Above CPU based on execution of the aforementioned sleep instruction by Above CPU.
[0011] In invention of the claim 2 above-mentioned publication, the aforementioned power management equipment has the feature of the system LSI which is invention

according to claim 3 in having made it the composition which resumes the clock supply to Above CPU in response to the end of processing of the aforementioned hardware module.

[0012] In invention of the claim 3 above-mentioned publication, the aforementioned sleep instruction has the 1st field which specifies by the end of processing of which hardware module the clock supply to Above CPU is resumed, and the feature of the system LSI which is invention according to claim 4 is in the aforementioned power management equipment for having carried out to the composition which resumes the clock supply to Above CPU with reference to the content of the 1st field of the above.

[0013] Having the 2nd field where the feature of the system LSI which is invention according to claim 5 specifies the register with which the aforementioned sleep instruction stored the value of the 1st field of the above in invention of the claim 4 above-mentioned publication, the aforementioned power management equipment is to have made it the composition in which the clock supply to Above CPU is resumed with reference to the content of the above 1st and the 2nd field.

[0014] The feature of the system LSI which is invention according to claim 6 is in the above-mentioned claim 2 or invention according to claim 5 to have made the routine in the aforementioned program used in order to wait for the end of processing of the aforementioned hardware module the composition automatically transposed to the aforementioned sleep instruction.

[0015] The feature of the power management method which is invention according to claim 7 To the sleep instruction which it performs [instruction] in CPU and stops supply of the clock to this CPU When the content which directs the conditions of the operating state of the hardware module for making supply of the clock to CPU resume is included and CPU executes the aforementioned sleep instruction It is in generating the signal which stops supply of the clock to CPU, after storing the aforementioned conditions in a register, and generating the signal which makes supply of the clock to CPU resume, when the operating state of a hardware module agrees with the aforementioned conditions stored in the aforementioned register.

[0016]

[Embodiments of the Invention] Hereafter, the gestalt of operation of this invention is explained with reference to a drawing.

[0017] [1st operation gestalt] drawing 1 is the block diagram showing the composition of the system LSI concerning the 1st operation gestalt of this invention.

[0018] The fundamental composition of this system LSI has the composition of having added the power management module 20 which had sleep and a function for carrying out the Wake rise for CPU11, in the same system LSI (drawing 20) as the former equipped with CPU11, the hardware modules M0-Mn of plurality (one [or]), and the memory 12 that stores a program and data.

[0019] although a clock is supplied to CPU11 through this power management module 20, a partial circuit with the need of always supplying the clock is resembled as usual, and a clock is supplied to it Furthermore, an addition and change of a function are added to CPU11 so that the sleep instruction for changing CPU11 into a sleep state can be decoded and executed in addition to the conventional instruction set.

[0020] In addition t the operation code OP showing a sleep instruction, the instruction

code of a sleep instruction gives the Wake rise module field WU and the condition field CO, as shown in drawing 2 . The Wake rise module field WU is the field which specifies whether the Wake rise of CPU11 is carried out, when processing of which hardware modules M0-Mn is completed (the hardware module specified to be this appearance is hereafter called Wake rise module). It enables it to specify one or more hardware modules to be the Wake rise module fields WU by giving the bit corresponding to each hardware module. The condition field CO will specify (it is hereafter called OR conditions) for whether the Wake rise of CPU11 is carried out, if either (it being hereafter called AND conditions) or those processings end whether the Wake rise of CPU11 will be carried out if all of those processings are completed when two or more modules in the Wake rise module field WU are specified.

[0021] For example, in a system LSI with five hardware modules M0, M1, M2, M3, and M4, when processing of both hardware modules M0 and M2 is completed, the sleep instruction in the case of carrying out the Wake rise of CPU11 comes to be shown in drawing 3 .

[0022] However, bit width of face of an instruction is made into 32 bits, and 1 bit is assigned to the condition field CO and, in the case of AND conditions, in the case of "0" and OR conditions, it is made into "1." Moreover, the Wake rise module field WU corresponded to hardware modules M0-M4 sequentially from the least significant bit, and when the bit corresponding to each hardware modules M0-M4 was "1", the hardware module should be specified as a Wake rise module.

[0023] A sleep instruction uses it in the program in the conventional system LSI, replacing with the routine (it is hereafter called the waiting routine for a hardware processing end) only for waiting for the end of processing of a hardware module. For example, suppose that there was a routine as shown in drawing 4 by the program of the conventional system LSI. Here, mfc is taken as the instruction which loads the content of the register for hardware module control to a general-purpose register. Moreover, by state's being the register which stores the operating state of a hardware module, and corresponding to hardware modules M0-M4 from a least significant bit, respectively, if a certain bit is "1", the hardware module corresponding to the bit shall express that processing is under execution.

[0024] This waiting routine for a hardware processing end performs the following processings. First, the content of $state is loaded to a general-purpose register r2. Next, only the operating state of hardware modules M1 and M3 is extracted by performing the AND operation of "r2" and "0x0005." Then, comparison of "r2" and "r0 (for the content, "0" is always a register)" is performed, and if both are not equal, it branches to the instruction which a label called Loop attached, i.e., the first mfc instruction. Since it is that CPU11 does not perform the next processing until processing of hardware modules M0 and M2 ends this, the sleep of CPU11 can be carried out in the meantime. Then, by replacing this routine by one sleep instruction of drawing 3 , while waiting for the processing end of hardware modules M0 and M2, the sleep of CPU11 is carried out. Power consumption is not only reducible, but program length can be shortened by this.

[0025] Drawing 5 is the block diagram showing the composition of the power management module 20. This power management module 20 consists of a comparator

21, the Wake rise module register 22, a m dule state flag register 23, a sleep control flip-flop 24, and the OR gate 25.

[0026] The sleep control flip-flop 24 is a flip-flop for controlling supply of the clock to CPU11, and reset-signal RTF is inputted for the set signal STF from a comparator 21 from CPU11 again.

[0027] The input of the OR gate 25 is the clock creatine kinase (it is hereafter called a main clock) currently supplied to CPU11 with the output OUT of the sleep control flip-flop 24, and the composition of the conventional system LS 1, and an output serves as the clock CLK supplied to CPU11. The module state flag register 23 is a register for supervising the operating state of a hardware module, and is giving the bit corresponding to each hardware module. CPU11 sets the value of this register and each hardware module resets it. When CPU11 of the conventional system LSI has the register which has an equivalent function, it can substitute for this register. The Wake rise module register 22 is a register which stores the value of the Wake rise module field WU in sleep instruction code, and when CPU11 decodes / executes a sleep instruction, it transmits a storing value. A comparator 21 is a circuit for detecting the timing which carries out the Wake rise of CPU11. The value of the module state flag register 23, the value of the Wake rise module register 22, the value of the condition field CO in sleep instruction code, and interrupt signal BR are inputted, and reset-signal RTF is outputted to the sleep control flip-flop 24 with those values.

[0028] Operation of the system LSI of this operation gestalt is explained.

[0029] When an initial state, i.e., a system LSI, is reset, interrupt signal BR (reset interruption) is inputted into a comparator 21. Since the sleep control flip-flop 24 is reset by this and the output OUT of the sleep control flip-flop 24 is set to "0" by it, Clock CLK is supplied to CPU11. CPU11 sets the bit corresponding to the hardware module in the module state flag register 23 at the same time it gives a processing instruction to a certain hardware module.

[0030] Next, execution of a sleep instruction transmits the value of the Wake rise module field WU in sleep instruction code, and the value of the condition field CO to the Wake rise module register 22 and comparator 21 of the power management module 20, respectively. Moreover, the sleep control flip-flop 24 is set and the output OUT is set to "1."

[0031] If the output OUT of the sleep control flip-flop 24 which is one of the inputs of the OR gate 25 is set to "1", the output CLK of the OR gate 25, i.e., the clock signal to CPU11, will not be concerned with the value of the main clock creatine kinase which is another input, but it will be set to "1." That is, since it means that supply of the clock CLK to CPU11 was stopped, CPU11 will be in a sleep state.

[0032] Each hardware modules M0-M4 reset the bit corresponding to the hardware module of the module state flag register 23, after the given processing is completed. The comparator 21 is always comparing the value of the module state flag register 23 and the Wake rise module register 22, and when the result satisfies the conditions given in the condition field CO of sleep instruction code, it outputs reset-signal RTF to the sleep control flip-flop 24. Moreover, when interrupt signal BR is inputted, it is not concerned with a comparison result but reset-signal RTF is outputted.

[0033] If the sleep control flip-flop 24 is reset, since the output OUT of this flip-flop 24 will be set to "0", the main clock creatine kinase which is another input of the OR gate

25 is transmitted to the output CLK of the OR gate 25 as it is. Supply of the clock CLK to CPU11 is resumed by this, and CPU11 carries out the Wake rise by it.

[0034] The system LSI which carried three hardware modules M0, M1, and M2 as shown in drawing 6 is made into an example, and above-mentioned operation is explained still more concretely.

[0035] Drawing 7 is the circuit diagram showing the example of composition of the comparator 21 in the power management module 20 in this example. each bit of the module state flag register 23 here corresponding to hardware modules M0, M1, and M2, and the Wake rise module register 22 -- respectively (S0, S1, S2) -- and (W0, W1, W2) -- ** -- it carries out It is shown that the hardware module corresponding to them is processing performing (S0, S1, S2) when a value is "1", and (W0, W1, W2) shall show that the hardware module corresponding to them was specified as a Wake rise module, when a value is "1."

[0036] If a processing instruction is taken out from CPU11 to a hardware module M1, it will become = (S0, S1, S2) (1, 0, 0). Moreover, if a processing instruction is also given to a hardware module M2 while the hardware module M0 is performing processing, it will become = (S0, S1, S2) (1, 0, 1). Then, if the sleep instruction of drawing 8 to which the sleep of CPU11 is carried out till the processing end of both hardware modules M0 and M2 is decoded / executed, "0" as which (1, 0, 1) express AND conditions in a comparator 21 to (W0, W1, W2) will be transmitted first, respectively.

[0037] Next, CPU11 will be in a sleep state by inputting the set signal STF into the sleep control flip-flop 24. Within the comparator 21, the value of (S0, S1, S2), and (W0, W1, W2) is compared by the combinational circuit (AND condition block 21a, OR condition block 21b). Since AND conditions are specified by the sleep instruction, the output of AND condition block 21a is chosen by multiplexer 21c.

[0038] In AND condition block 21a, the bit corresponding to the hardware modules M0, M1, and M2 of the module state flag register 23 and the Wake rise module register 22 is inputted into NAND gates AC0, AC1, and AC2, respectively. (S0, S1, S2) The output of AND-gate AC3 to which these values are considered as an input at the time of = (1, 0, 1) and = (W0, W1, W2) (1, 0, 1) since the outputs of NAND gates AC0, AC1, and AC2 are "0", "1", and "0", respectively is set to "0." Since it is W1=0, it is not concerned with the value of S1, i.e., the operating state of a hardware module M1, but the output of NAND gate AC1 is always set to "1." That is, only the operating state of the hardware module specified as a Wake rise module influences the output of AND-gate AC3.

[0039] Since the output of NAND gate AC2 is "0" even if processing of a hardware module M0 is completed, it is set to S0=0 and the output of NAND gate AC0 is set to "1", the output of AND-gate AC3 is still "0." If processing of a hardware module M2 is also ended and it is set to S2=0, the output of NAND gate AC2 will be set to "1." NAND gates AC0, AC1, and AC2 -- if all outputs are set to "1", the output of AND-gate AC3 will be set to "1" Since the output of AND-gate AC3 is chosen by multiplexer 21c, the output of multiplexer 21c is set to "1." Since it is the output of multiplexer 21c, and the input whose interrupt signal BR is 21d of OR gates, when interrupt signal BR is not inputted, the output of AND-gate AC3 is transmitted to the output which is 21d of OR gates as it is. Since this becomes the output of a comparator 21, i.e., reset-signal RTF, when processing of both hardware modules M0 and M2 is completed, reset-signal RTF

is set to "1." If the sleep control flip-flop 24 is reset by this reset-signal RTF and the output OUT is set to "0" by it, the value of the main clock creatine kinase will be transmitted to the output of the OR gate 25 as it is, and supply of the clock CLK to CPU11 will be resumed.

[0040] Next, the case where the sleep instruction of drawing 9 to which the sleep of CPU11 is carried out till the end of a hardware module M0 or processing of M2 is executed is explained.

[0041] Since OR conditions are specified by the sleep instruction, the output of OR condition block 21b is chosen by multiplexer 21c. In OR condition block 21b, the bit corresponding to M0, M1, and M2 of the module state flag register 23 and the Wake rise module register 22 is inputted into the AND gates OC0, OC1, and OC2, respectively. However, the value by which the value of the module state flag register 23 was reversed is inputted. (S0, S1, S2) The output of OR-gate OC3 to which these values are considered as an input at the time of = (1, 0, 1) and = (W0, W1, W2) (1, 0, 1) since the outputs of the AND gates OC0, OC1, and OC2 are "0", "0", and "0", respectively is set to "0." since it is W1=0 -- the value of S1, i.e., the operating state of a hardware module M1, -- not being concerned -- the output of AND-gate OC1 -- "0" -- always -- ** -- it becomes That is, only the operating state of the hardware module specified as a Wake rise module influences the output of OR-gate OC3.

[0042] If processing of a hardware module M0 is completed and it is set to S0=0, the output of AND-gate OC0 will be set to "1", and the output of OR-gate OC3 will also be set to "1" by this. This value lets multiplexer 21c and 21d of OR gates pass, and is outputted to the sleep control flip-flop 24 as a reset-signal RTF.

[0043] When interrupt signal BR is inputted, regardless of the output of multiplexer 21c, the output of 21d of OR gates is set to "1", and reset-signal RTF will be outputted to the sleep control flip-flop 24.

[0044] Here, although only AND conditions and OR conditions explained composition and operation for the conditions which carry out the Wake rise of CPU11, the circuit of a condition block is added and other condition logic can be realized by using two or more bits for the condition field CO of sleep instruction code.

[0045] Moreover, even if it carries out the Wake rise of CPU11 or processing of which hardware module is completed by the processing end of a specific hardware module, in carrying out the Wake rise of CPU11, a comparator 21, the Wake rise module register 22, and the module state flag register 23 become unnecessary. What is necessary is just to input into the sleep control flip-flop 24 by setting the result of the OR operation of the processing terminate signal from a hardware module, and interrupt signal BR to reset-signal RTF. The power management module 20 can be simplified by this.

[0046] When interrupt signal BR was inputted and it changes into the state where there is a circuit, using reset-signal RTF being outputted to the sleep control flip-flop 24, it becomes possible to carry out the Wake rise of CPU11. For example, what is necessary is just to make it output interrupt signal BR, when a data bus changed into a predetermined state, the Wake rise of CPU11 was carried out, the circuit which supervises the state of a data bus is added and a data bus changes into the state.

[0047] Although the composition which added the power management module 20 to the conventional system LSI explained with this operation form, this is for giving explanation easy and may include this function into CPU11. In this case, it is that a

clock is always supplied to the circuit equivalent to the Wake rise module register 22, the module state flag register 23, and the sleep control flip-flop 24, and same operation can be realized.

[0048] Thus, with this operation form, since the timing to which the sleep of CPU11 is carried out was controlled by software (sleep instruction), it can determine when a programmer carries out the sleep of CPU11, and it becomes possible to avoid the degradation by execution of useless sleep. Furthermore, only a suitable period becomes possible [carrying out sleep] by controlling the Wake rise of CPU11 by hardware, without predicting a sleep period. It becomes possible to raise a low-power-ized effect sharply, this avoiding the degradation by execution of useless sleep. Moreover, since the power management module 20 which realizes these functions can be mounted by very easy hardware, its overhead of the area by this and power consumption is also small.

[0049] Since the waiting routine for a hardware module processing end which consisted of two or more instructions is transposed to one sleep instruction, the amount of memory which is needed since a program is stored becomes small, and the number of instructions performed also becomes less than the conventional program. Furthermore, since instruction memory is not accessed while CPU is carrying out sleep, the sleep also of the instruction memory can be carried out in the meantime. The power consumption of instruction memory is also cut down by these things.

[0050] The 2nd operation form of the [2nd operation form] this invention is explained with reference to drawing 10 , drawing 11 , and drawing 12 .

[0051] In the system LSI shown in drawing 1 with five hardware modules M0, M1, M2, M3, and M4, the case where it is expressed with a flow chart as a part of program performed shows to drawing 10 is considered. Here, suppose that a processing instruction is taken out with processing 1.1, processing 1.2, processing 2.1, processing 2.2, and processing 3 to hardware modules M0, M1, M2, M3, and M4, respectively. Since the waiting routine for a hardware module processing end is after processing 3 is completed, by transposing this to a sleep instruction, the sleep of CPU11 is carried out and curtailment of power consumption is aimed at.

[0052] However, supposing it waits to complete processing of all hardware modules in which this routine was able to give the processing instruction in the performed processing, in the sleep instruction code of drawing 2 , it is unrealizable. In the sleep instruction of drawing 2 , since conditional branching occurs by this program to presuming the Wake rise instruction clearly, it is because the hardware module which has a processing instruction issued is not determined as 1 set.

[0053] Then, the 2nd sleep instruction code shown in drawing 11 is introduced. This sleep instruction code consists of an operation code OP and the one register field RG. One certain register is specified in the register field RG. The contents equivalent to the value of the Wake rise module field WU in the sleep instruction code of drawing 2 and the condition field CO are stored in the register specified here. When decoding and executing this sleep instruction, the Wake rise of CPU11 is controlled by transmitting the value of the register directed in the register field RG to the power management module 20.

[0054] Next, how to change CPU11 into a sleep state is explained with reference to the flow chart of drawing 12 using this sleep instruction.
[0055] The most significant bit of the register specified in the register field RG presupposes that it is equivalent to the condition field CO, and 5 bits is equivalent to the Wake rise module field WU from a least significant bit. Moreover, suppose that a general-purpose register r8 is specified in a sleep instruction.
[0056] In drawing 12 , first, before performing processing in connection with sleep operation, it initializes by substituting "0x0000" for "r8." After giving a processing instruction to a hardware module M0 in processing 1.1, an OR operation is performed for the value of "r8", and "0x0001." Similarly, after giving a processing instruction to hardware modules M1, M2, M3, and M4 in processing 1.2, processing 2.1, processing 2.2, and processing 3, the value of "r8" and the OR operation of "0x0002", "0x0004", "0x0008", and "0x0010" are performed, respectively, and the result is stored in "r8." If it is AND conditions and is "0x0000" and OR conditions before executing a sleep instruction in order to specify the value of the condition field CO, the OR operation of "0x8000" and "r8" is performed, respectively.
[0057] When performing processing 1.1 and processing 2.1 and carrying out the Wake rise of CPU11 on AND conditions by doing in this way, the contents of "r8" are set to "0x0015", and sleep operation of carrying out the Wake rise of CPU11 according to the AND conditions of hardware modules M0 and M2 can be realized.
[0058] In addition, since pipeline processing is carried out, there is a case where he wants to make the instruction preceded before a sleep instruction goes into the execution stage completed. For example, when accessing memory 12 by processing by CPU11 and loading/store instruction is left with the access demand from CPU11 coming out, it is the case where it becomes impossible ATASESU [other modules]. In this case, a number required before a sleep instruction of NOP instructions are inserted, and a sleep instruction is made to execute after the instruction is completed.
[0059] the [next, / the above 1st and] -- the sleep to CPU11 used for 2 operation forms and the order of the real way of the Wake rise processing are explained with reference to the flow chart of drawing 13
[0060] First, CPU11 decodes the instruction executed next (Step S11), and it judges whether it is the sleep instruction to which the sleep of CPU11 is carried out (Step S12). If it is a sleep instruction, the conditions for carrying out the Wake rise of CPU11 directed by this instruction are stored in a register (Step S13). Furthermore, the signal which stops supply of the clock to CPU11 is generated, and CPU11 carries out sleep with this signal (Step S14).
[0061] While CPU11 is carrying out sleep, after the processing under execution of a certain hardware module is completed, a processing terminate signal is generated (Step S15). It is judged whether the conditions which make the clock supply to stored CPU11 resume were fulfilled (Step S16), if filled by the processing end of the hardware module, the signal which makes supply of the clock to CPU11 resume will be generated by it, and CPU11 will carry out the Wake rise by it (Step S17).
[0062] the [the [3rd operation form] above 1st and] -- with 2 operation forms, when CPU11 cannot perform the next processing until processing of a certain hardware module is completed, CPU11 is changed into a sleep state with software (sleep

instruction), and the Wake rise of CPU11 is carried out by the processing terminate signal from the hardware module In this case, the operation code OP to which the sleep instruction c de used expresses a sleep instruction, The Wake rise module field WU which specifies by the processing terminate signal from which hardware module the Wake rise of CPU11 is carried out, When two or more hardware modules in the Wake rise module field WU are specified, those AND conditions that will carry out the Wake rise of CPU11 if processing is all completed, And if ones of those processings is completed, it will consist of the condition field CO showing OR conditions which carry out the Wake rise of CPU11. the [the above 1st and] -- with 2 operation forms, since such sleep instruction code was ungenerable with a compiler, there was fault that the waiting routine for a hardware module processing end had to be transposed to a sleep instruction by the help

[0063] In order to cancel this fault, with the **** 3 operation gestalt, the function which transposes automatically the waiting routine for a hardware module processing end in the conventional program to the aforementioned sleep instruction is prepared. Hereafter, the technique is explained in detail.

[0064] Drawing 14 is drawing showing an example of the waiting routine for a hardware module processing end in the program of a system LSI.

[0065] In this drawing, the value of status-register state of a hardware module is first loaded to a general-purpose register r2 by mfc instruction. Here, each bit of status-register state will be set to "0", if a corresponding hardware module is performing processing and "1" and processing will be ended.

[0066] Then, a mask is covered over the value loaded by andi instruction. Since the 3rd operand of an andi instruction is "0x0003", this has taken out only the operating state of the hardware module (hereafter referred to as M0 and M1, respectively) corresponding to the 0th bit (LSB) of status-register state, and the 1st bit. And a bne instruction performs comparison with the register r0 which is always "0", and if the result is not equal, it already branches to the degree of -, and mfc2 instruction, and if equal, it will escape from the routine. That is, it will be called the routine for which it waits that processing of both hardware modules M0 and M1 ends this routine.

[0067] Next, it considers transposing this routine to a sleep instruction.

[0068] Since the Wake rise of CPU11 is carried out by the end of processing of hardware modules M0 and M1 in this example, the Wake rise module field WU of a sleep instruction is set to "0x0003." Moreover, if processing of all the hardware modules specified at both hardware modules M1 [M0 and] WU, i.e., the Wake rise module field, is completed, since the Wake rise of CPU11 will be carried out, the value of the condition field CO serves as AND conditions. Therefore, a corresponding sleep instruction comes to be shown in drawing 15 . Here, as for the value of "111111" and the condition field CO, "0" and OR conditions set [AND conditions] the operation code OP showing a sleep instruction to "1."

[0069] Another example of the waiting routine for a hardware module processing end is shown in drawing 16 . In this example, first, "0x0003" is substituted for the register r3, and this is used as a mask by the next and instruction of mfc2 instruction. When the value of status-register state over which the mask was covered is the same as r3 (i.e., when having ended neither of hardware modules M0 and M1 of the processings),

branching is materialized, and when it differs (i.e., when one processing of the hardware m dules M0 or M1 is completed), it escapes from this routine. That is, this routine is a routine f r waiting to complete one pr cessing of the hardware modules M0 r M1. Therefore, when this routine is transposed to a sleep instruction, it comes to be shown in drawing 17 . Here, the value of the Wake rise module field WU is "0x0003" which shows hardware modules M0 and M1, and since the condition field CO carries out the sleep of CPU11 until one processing of the hardware modules M0 or M1 is completed, it serves as OR conditions.

[0070] In a compiler, a sleep instruction is automatically generable by transposing the procedure which generates such an instruction sequence to the procedure which generates a sleep instruction.

[0071] Next, the procedure for transposing the waiting routine for a hardware module processing end to a sleep instruction is explained with reference to drawing 18 and drawing 19 .

[0072] In the flow chart of drawing 18 , a source code is first changed into an assembly code with a compiler (Step S21). Next, a hardware module processing end **** routine is changed into a sleep instruction by filtering this (Step S22). When the changed assembly code assembles at the end, a machine language including a sleep instruction is generated (Step S23).

[0073] A translation table is used for filtering at Step S22. The sleep instruction corresponding to the instruction sequence which has realized the waiting routine for a hardware module processing end, and its instruction sequence constructs a translation table, and it consists of ******. A constant and a variable express the operand of an instruction sequence. The example of the translation table for changing the instruction sequence of drawing 14 and drawing 15 into a sleep instruction at drawing 19 is shown.

[0074] In drawing 19 , a declaration of a variable and a constant is made first. Here, state below $const, and r0, and and or are constants, and reg1, reg2, imm1, and imm2 below $var are a variable. LABEL1 and LABE2 below $label are a special variable for the judgment with a branching place in agreement (it is hereafter called a label variable). The portion surrounded by $table and $endtab1e is the actual condition of a translation table, and below $stream in this is the format of the sleep instruction which an instruction sequence and below $format change.

[0075] Furthermore, the procedure of filtering is explained. The filter performs comparison with the instruction sequence of a translation table sequentially from the beginning of an assembly code. Comparison is performed as follows. First, it judges whether an operation code OP is in agreement. If in agreement, it will judge whether next an operand is in agreement. In this case, the operand given as a constant by the translation table compares as it is. When substituting the value of the operand of an assembly code in the case of the first comparison performed using the variable and performing comparison with the variable next at it, it compares with the operand given as a variable using the substituted value. If both are in agreement as a result of comparison, the instruction sequence in an assembly code will be replaced according to the format of the sleep instruction to which it was given by $format.

[0076] When there is a hardware module processor-limited routine as shown in a certain acene pulley code at drawing 14 hereafter, the procedure at the time of changing this into a sleep instruction by the translation table shown in drawing 19 is explained concretely.
[0077] Since "LABEL1:" appeared in the assembly code, the position of this label is stored. Next, since there is mfc2 instruction and this is in agreement with the instruction of the beginning of the instruction sequence in a translation table, the operand is compared. Since the first operand of the mfc2 instruction in a translation table is a variable of reg1, the value r2 of the first operand of the mfc2 instruction in an assembly code is substituted for this. Since the second operand of mfc2 instruction of a translation table is a constant, it compares as it is.
[0078] Since the mfc instruction of an operand corresponds, the next instruction is compared. Since both are in agreement by andi instruction, a first operand is compared. Since the first operand of an andi instruction of a translation table is a variable reg2, the first operand r2 of the andi instruction in an assembly code is substituted for this.
[0079] Since the second operand of an andi instruction of a translation table is reg1 of existing appearance, it is compared using r2 previously substituted for this variable. "0x0003" is substituted for imm1 by comparison of the 3rd operand. In comparison of the next bne instruction, a second operand is the same.
[0080] Since LABEL1 of the 3rd operand of a bne instruction of a translation table is a label variable, it judges whether the label and branching place of the 3rd operand of the bne instruction in an assembly code are the same. Since both sides have branched to the mfc2 instruction in the case of this example, it is in agreement. Since it was altogether in agreement with the instruction sequence of a translation table, the instruction sequence in an assembly code is transposed to a sleep instruction according to the format to which it was given by $format. Since imm1 is a variable in the case of this example, "0x0003" substituted at the time of comparison of an andi instruction is used.
[0081] Thus, with this operation gestalt, since the instruction sequence which was a waiting routine for a hardware module processing end in the conventional program can be automatically transposed to a sleep instruction, the effort of the part and a programmer is decreased and it becomes possible to attain the increase in efficiency of the low-power design in a system LSI.
[0082]
[Effect of the Invention] It becomes possible to attain low-power-ization, avoiding the degradation by execution of useless sleep according to the system LSI which is invention according to claim 1 to 4, since a halt (sleep) of the clock supply to CPU is controllable by software (sleep instruction), as explained to the detail above. Furthermore, it becomes possible by controlling resumption (Wake rise) of the clock supply to CPU by hardware to raise a low-power-ized effect sharply, being able to carry out the sleep only of the suitable period and avoiding the degradation by execution of useless sleep without predicting a sleep period. Moreover, since the power management equipment which realizes these functions can be mounted by very easy hardware, its overhead of the area by this and power consumption is also small. Moreover, since the real line count of an instruction becomes less than the conventional program by transposing the waiting

routine for a hardware module processing end which exists in the conventional program to one sleep instruction, the power consumption by memory access is also reducible. Furthermore, since two or more instructions are transposed to one instruction, the amount of memory which is needed since a program is stored also becomes small.

[0083] According to the system LSI which is invention according to claim 5, a sleep instruction

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号
特開2000－112585
（P2000－112585A）
(43)公開日　平成12年４月21日(2000.4.21)

(51)Int.Cl.'		識別記号	FI			テーマコード'(参考)
G06F	1/32		G06F	1/00	332Z	5B011
	1/26			1/04	301C	5B079
	1/04	301		1/00	330C	

審査請求　未請求　請求項の数９　ＯＬ　（全 13 頁）

(21)出願番号　特願平10－281610

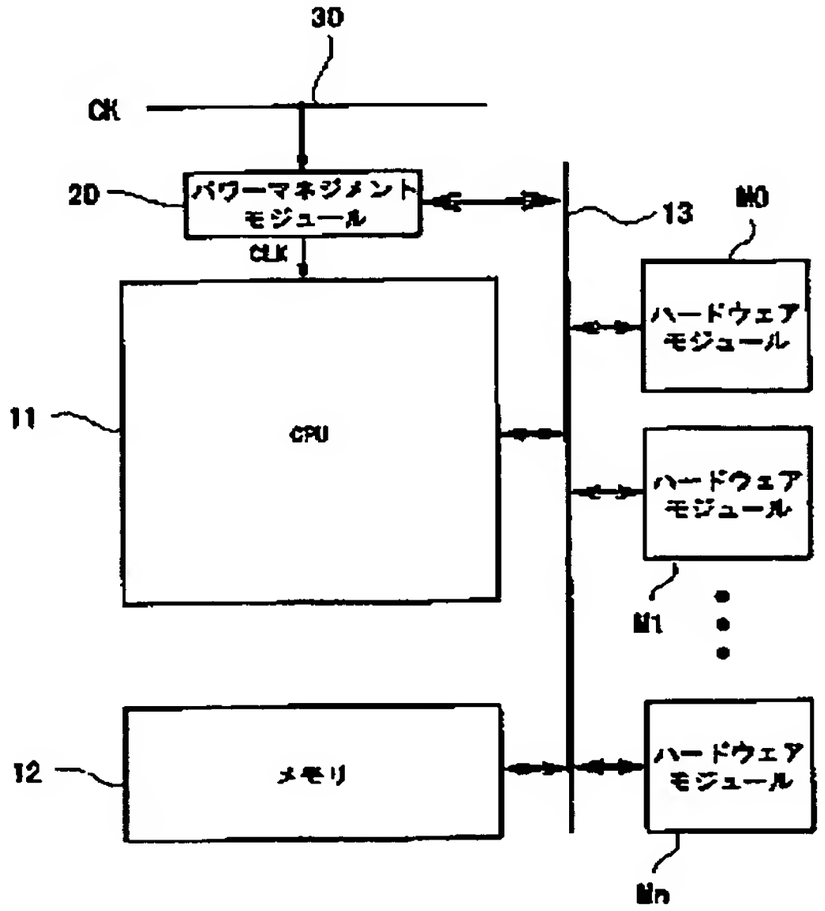
(22)出願日　平成10年10月２日(1998.10.2)

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　河邉　直之
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内
(72)発明者　宇佐美　公良
神奈川県川崎市幸区堀川町580番１号　株式会社東芝半導体システム技術センター内
(74)代理人　100083806
弁理士　三好　秀和　（外７名）

Ｆターム(参考)　5B011 EA08 EB00 LL12
5B079 AA07 BA12 BB02 BC01

(54)【発明の名称】　システムＬＳＩ及びパワーマネジメント方法

(57)【要約】
【課題】　ＣＰＵの無駄なスリープの実行による性能低下を避けつつ低消費電力化効果を大幅に向上させることができるシステムＬＳＩを提供する。
【解決手段】　所定の処理を実行する１つまたは複数のハードウェアモジュールと、前記ハードウェアモジュールの動作をプログラムに従って制御するＣＰＵとを備えたシステムＬＳＩにおいて、前記ＣＰＵに対するクロック供給の停止及び再開を制御するパワーマネジメント装置を設ける。すなわち、ＣＰＵに対するクロック供給の停止を行うためのスリープ命令を実行する手段をＣＰＵに設け、前記パワーマネジメント装置は、ＣＰＵによる前記スリープ命令の実行に基づいて、ＣＰＵに対するクロック供給を停止すると共に、記ハードウェアモジュールの処理の終了を受けて、ＣＰＵに対するクロック供給を再開する構成にする。



【特許請求の範囲】
【請求項１】　所定の処理を実行する１つまたは複数のハードウェアモジュールと．前記ハードウェアモジュールの動作をプログラムに従って制御するＣＰＵとを備えたシステムＬＳＩにおいて．
前記ＣＰＵに対するクロック供給の停止及び再開を制御するパワーマネジメント装置を設けたことを特徴とするシステムＬＳＩ。
【請求項２】　前記ＣＰＵに対するクロック供給の停止を行うためのスリープ命令を実行する手段を前記ＣＰＵに設け、
前記パワーマネジメント装置は、前記ＣＰＵによる前記スリープ命令の実行に基づいて、前記ＣＰＵに対するクロック供給を停止する構成にしたことを特徴とする請求項１記載のシステムＬＳＩ。
【請求項３】　前記パワーマネジメント装置は．前記ハードウェアモジュールの処理の終了を受けて、前記ＣＰＵに対するクロック供給を再開する構成にしたことを特徴とする請求項２記載のシステムＬＳＩ。
【請求項４】　前記スリープ命令は．どのハードウェアモジュールの処理の終了によって前記ＣＰＵに対するクロック供給を再開するかを指定する第１のフィールドを有し．
前記パワーマネジメント装置は、前記第１のフィールドの内容を参照して前記ＣＰＵに対するクロック供給を再開する構成にしたことを特徴とする請求項３記載のシステムＬＳＩ。
【請求項５】　前記スリープ命令は．前記第１のフィールドの値を格納したレジスタを指定する第２のフィールドを有し、
前記パワーマネジメント装置は、前記第１と第２のフィールドの内容を参照して前記ＣＰＵに対するクロック供給を再開する構成にしたことを特徴とする請求項４記載のシステムＬＳＩ。
【請求項６】　前記ハードウェアモジュールの処理の終了を待つために用いられる前記プログラム中のルーチンを前記スリープ命令に自動的に置き換える構成にしたことを特徴とする請求項２乃至請求項５記載のシステムＬＳＩ。
【請求項７】　前記ＣＰＵに接続されたメモリを有し、前記パワーマネジメント装置は、前記ＣＰＵへのクロック供給の停止時に前記メモリへのクロック供給を停止し．且つ前記ＣＰＵへのクロック供給の再開時に前記メモリへのクロック供給を再開する構成にしたことを特徴とする請求項１乃至請求項６記載のシステムＬＳＩ。
【請求項８】　前記メモリは、前記ＣＰＵ、前記ハードウェアモジュール及び前記パワーマネジメント装置と同一のチップ上に形成したことを特徴とする請求項７記載のシステムＬＳＩ。
【請求項９】　ＣＰＵにおいて実行され該ＣＰＵへのクロックの供給を停止させるスリープ命令に、ＣＰＵへのクロックの供給を再開させるためのハードウェアモジュールの動作状態の条件を指示する内容を含めておき、
ＣＰＵが前記スリープ命令を実行したときに、前記条件をレジスタに格納してからＣＰＵへのクロックの供給を停止させる信号を生成し．
ハードウェアモジュールの動作状態が．前記レジスタに格納された前記条件と合致したときに．ＣＰＵへのクロックの供給を再開させる信号を生成することを特徴とするパワーマネジメント方法。

【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は．チップ上にＣＰＵと１つまたは複数のハードウェアモジュールを搭載したシステムＬＳＩ．及びその低消費電力化を実現するパワーマネジメント方法に関する。
【０００２】
【従来の技術】一般に、従来のシステムＬＳＩは．図２０に示すように．チップ上にＣＰＵ１１１と、複数のハードウェアモジュールＭ０．Ｍ１，…、Ｍｎと．命令やデータを格納するメモリ１１２とを搭載した構成となっている。ＣＰＵ１１１は．それ自身が行う処理のほかに．ハードウェアモジュールＭ０～Ｍｎに対して処理命令を出す。ハードウェアモジュールＭ０～Ｍｎの動作状態．つまり与えられた処理を実行中であるか、与えられた処理の実行が終了して次の処理命令の待機中であるか．あるいはスリープ中であるかといった動作状態は、ＣＰＵ１１１によって監視されている。
【０００３】かかるシステムＬＳＩの低消費電力化に関しては、例えば次のような技術が提案されている。
（１）あるハードウェアモジュールＭ０～Ｍｎが動作していなくてもよい期間には．ＣＰＵ１１１は、そのハードウェアモジュールＭ０～Ｍｎへのクロックの供給を停止して、該ハードウェアモジュールＭ０～Ｍｎをスリープ状態にすることにより．低消費電力化を図る。（２）ハードウェアモジュールＭ０～Ｍｎ自身が、与えられた処理を終了したら自動的にスリープ状態にすることによって．低消費電力化を図る。
【０００４】
【発明が解決しようとする課題】しかしながら．従来のシステムＬＳＩの低消費電力化に関しては、ＣＰＵによって消費される電力がチップ全体で消費される電力の多くを占めており．ＣＰＵの低消費電力化が重要な課題となっている。
【０００５】このような観点から、ＣＰＵ１１１をスリープ状態にする手法として．次のようなことが考えられる。ＣＰＵ１１１をスリープ状態にすることができる期間としては、例えば、ハードウェアモジュールＭ０～Ｍｎが行った処理の結果を用いてＣＰＵ１１１が次の処理を行う場合などのように．データ依存などの関係から、

ハードウェアモジュールＭ０～Ｍｎが所定の与えられた処理を終了するまでＣＰＵ１１１が次の処理を実行できない期間がある。そのようなハードウェアモジュールＭ０～Ｍｎの処理終了までの待ち時間の間はＣＰＵ１１１をスリープ状態にすることができるが．その方法が問題となる。

【０００６】ハードウェア制御によってＣＰＵ１１１をスリープ状態にする場合では、無駄なスリープの実行による性能低下や．消費電力削減効果の低下を招く恐れがある。なぜなら．ＣＰＵ１１１が次に行うべき動作を予測できないために、ＣＰＵ１１１をスリープ状態にした直後にウェイクアップしなければならなかったり．ＣＰＵ１１１をスリープ状態にすることができるのにスリープさせなかったりすることがあるからである。また、ハードウェアでスリープさせるタイミングを厳密に制御しようとすると．そのハードウェアによって面積及び消費電力のオーバヘッドか大きくなるという問題もある。

【０００７】一方、ソフトウェア制御によってＣＰＵ１１１をスリープ及びウェイクアップする場合は．ＣＰＵ１１１をスリープ状態にする期間を予め見積もり．その期間をスリープ命令によって指示してスリープ状態にした後にウェイクアップする。しかし．スリープ期間を多く見積もると．無駄な期間にスリープすることになり、結局．性能低下を招くことになる。逆に、少なく見積もってしまうと．実際はスリープしていてよい期間にクロックを供給することになり．消費電力削減効果が低下するという問題がある。

【０００８】本発明は、上述の如き従来の問題点を解決するためになされたもので．その目的は、ＣＰＵの無駄なスリープの実行による性能低下を避けつつ低消費電力化効果を大幅に向上させることができるシステムＬＳＩ及びパワーマネジメント方法を提供することである。またその他の目的は、システムＬＳＩにおける低消費電力設計の効率化を図ることができるシステムＬＳＩを提供することである。

【０００９】

【課題を解決するための手段】上記目的を達成するために．請求項１記載の発明であるシステムＬＳＩの特徴は．所定の処理を実行する１つまたは複数のハードウェアモジュールと．前記ハードウェアモジュールの動作をプログラムに従って制御するＣＰＵとを備えたシステムＬＳＩにおいて．前記ＣＰＵに対するクロック供給の停止及び再開を制御するパワーマネジメント装置を設けたことにある。

【００１０】請求項２記載の発明であるシステムＬＳＩの特徴は、上記請求項１記載の発明において、前記ＣＰＵに対するクロック供給の停止を行うためのスリープ命令を実行する手段を前記ＣＰＵに設け．前記パワーマネジメント装置は．前記ＣＰＵによる前記スリープ命令の実行に基づいて．前記ＣＰＵに対するクロック供給を停止する構成にしたことにある。

【００１１】請求項３記載の発明であるシステムＬＳＩの特徴は、上記請求項２記載の発明において、前記パワーマネジメント装置は、前記ハードウェアモジュールの処理の終了を受けて、前記ＣＰＵに対するクロック供給を再開する構成にしたことにある。

【００１２】請求項４記載の発明であるシステムＬＳＩの特徴は、上記請求項３記載の発明において、前記スリープ命令は、どのハードウェアモジュールの処理の終了によって前記ＣＰＵに対するクロック供給を再開するかを指定する第１のフィールドを有し．前記パワーマネジメント装置は．前記第１のフィールドの内容を参照して前記ＣＰＵに対するクロック供給を再開する構成にしたことにある。

【００１３】請求項５記載の発明であるシステムＬＳＩの特徴は、上記請求項４記載の発明において、前記スリープ命令は、前記第１のフィールドの値を格納したレジスタを指定する第２のフィールドを有し、前記パワーマネジメント装置は、前記第１と第２のフィールドの内容を参照して前記ＣＰＵに対するクロック供給を再開する構成にしたことにある。

【００１４】請求項６記載の発明であるシステムＬＳＩの特徴は、上記請求項２乃至請求項５記載の発明において．前記ハードウェアモジュールの処理の終了を待つために用いられる前記プログラム中のルーチンを前記スリープ命令に自動的に置き換える構成にしたことにある。

【００１５】請求項７記載の発明であるパワーマネジメント方法の特徴は、ＣＰＵにおいて実行され該ＣＰＵへのクロックの供給を停止させるスリープ命令に．ＣＰＵへのクロックの供給を再開させるためのハードウェアモジュールの動作状態の条件を指示する内容を含めておき．ＣＰＵが前記スリープ命令を実行したときに．前記条件をレジスタに格納してからＣＰＵへのクロックの供給を停止させる信号を生成し、ハードウェアモジュールの動作状態が．前記レジスタに格納された前記条件と合致したときに．ＣＰＵへのクロックの供給を再開させる信号を生成することにある。

【００１６】

【発明の実施の形態】以下．図面を参照して本発明の実施の形態を説明する。

【００１７】［第１実施形態］図１は．本発明の第１実施形態に係るシステムＬＳＩの構成を示すブロック図である。

【００１８】このシステムＬＳＩの基本的な構成は、ＣＰＵ１１と、複数（または１つ）のハードウェアモジュールＭ０～Ｍｎと、プログラムやデータを格納するメモリ１２とを備えた従来と同様のシステムＬＳＩ（図２０）において．ＣＰＵ１１をスリープ及びウェイクアップさせるための機能を持ったパワーマネジメントモジュール２０を加えた構成となっている。

【００１９】ＣＰＵ１１には、このパワーマネジメントモジュール２０を介してクロックを供給するが、常にクロックを供給している必要のある部分回路には、従来通りにクロックが供給されるようになっている。さらに、ＣＰＵ１１には、従来の命令セットに加え、ＣＰＵ１１をスリープ状態にするためのスリープ命令をデコード且つ実行できるように、機能の追加及び変更が加えられる。

【００２０】スリープ命令の命令コードは、図２に示すように、スリープ命令を表すオペコードＯＰに加え、ウェイクアップモジュールフィールドＷＵと、条件フィールドＣＯとを持たせる。ウェイクアップモジュールフィールドＷＵは、どのハードウェアモジュールＭ０～Ｍｎの処理が終了したらＣＰＵ１１をウェイクアップさせるかを指定するフィールドである（以下、この様に指定されたハードウェアモジュールをウェイクアップモジュールと呼ぶ）。ウェイクアップモジュールフィールドＷＵに、各ハードウェアモジュールに対応したビットを持たせることで、１つまたは複数のハードウェアモジュールを指定することができるようにする。条件フィールドＣＯは、ウェイクアップモジュールフィールドＷＵで複数のモジュールが指定された場合に、それらの処理が全て終了したらＣＰＵ１１をウェイクアップさせるのか（以下、ＡＮＤ条件と呼ぶ）、あるいは、それらの処理のいずれかが終了したらＣＰＵ１１をウェイクアップさせるのか（以下、ＯＲ条件と呼ぶ）を指定する。

【００２１】例えば、５つのハードウェアモジュールＭ０、Ｍ１、Ｍ２、Ｍ３、Ｍ４を持つシステムＬＳＩにおいて、ハードウェアモジュールＭ０及びＭ２の両方の処理が終了したらＣＰＵ１１をウェイクアップさせる場合のスリープ命令は、図３に示すようになる。

【００２２】ただし、命令のビット幅を３２ビットとし、条件フィールドＣＯに１ビットを割当て、ＡＮＤ条件の場合は“０”、ＯＲ条件の場合は“１”とする。また、ウェイクアップモジュールフィールドＷＵは、最下位ビットから順に、ハードウェアモジュールＭ０～Ｍ４に対応しており、各ハードウェアモジュールＭ０～Ｍ４に対応するビットが“１”のときに、そのハードウェアモジュールがウェイクアップモジュールとして指定されたものとする。

【００２３】スリープ命令は、従来のシステムＬＳＩでのプログラムにおいて、ハードウェアモジュールの処理の終了を待つためだけのルーチン（以下、ハードウェア処理終了待ちルーチンと呼ぶ）と置き換えて使用する。例えば、従来のシステムＬＳＩのプログラムで図４に示すようなルーチンがあったとする。ここで、ｍｆｃはハードウェアモジュール制御用のレジスタの内容を汎用レジスタにロードする命令とする。また、ｓｔａｔｅはハードウェアモジュールの動作状態を格納するレジスタで、最下位ビットからそれぞれハードウェアモジュールＭ０～Ｍ４に対応するものとし、あるビットが“１”であれば、そのビットに対応するハードウェアモジュールが処理を実行中であることを表すものとする。

【００２４】このハードウェア処理終了待ちルーチンは、次のような処理を行うものである。まず、＄ｓｔａｔｅの内容を、汎用レジスタｒ２にロードする。次に、「ｒ２」と「０ｘ０００５」のＡＮＤ演算を行うことによって、ハードウェアモジュールＭ１とＭ３の動作状態のみを抽出する。その後、「ｒ２」と「ｒ０（常に内容が“０”のレジスタ）」の比較を行い、両者が等しくなければ、Ｌｏｏｐというラベルのついた命令、つまり、最初のｍｆｃ命令に分岐する。これは、ハードウェアモジュールＭ０とＭ２の処理が終了するまで、ＣＰＵ１１は次の処理は行わない、ということであるから、この間は、ＣＰＵ１１はスリープすることができる。そこで、このルーチンを図３のスリープ命令１つで置き換えることにより、ハードウェアモジュールＭ０，Ｍ２の処理終了を待つ間はＣＰＵ１１をスリープさせる。これによって、消費電力を削減できるだけでなく、プログラム長も短縮することができる。

【００２５】図５は、パワーマネジメントモジュール２０の構成を示すブロック図である。このパワーマネジメントモジュール２０は、コンパレータ２１、ウェイクアップモジュールレジスタ２２、モジュール状態フラグレジスタ２３、スリープ制御フリップフロップ２４、及びＯＲゲート２５で構成されている。

【００２６】スリープ制御フリップフロップ２４は、ＣＰＵ１１へのクロックの供給を制御するためのフリップフロップであり、ＣＰＵ１１からセット信号ＳＴＦが、またコンパレータ２１からリセット信号ＲＴＦが入力される。

【００２７】ＯＲゲート２５の入力は、スリープ制御フリップフロップ２４の出力ＯＵＴと、従来のシステムＬＳＩの構成でＣＰＵ１１に供給されていたクロックＣＫ（以下、メインクロックと呼ぶ）であり、出力はＣＰＵ１１に供給されるクロックＣＬＫとなる。モジュール状態フラグレジスタ２３は、ハードウェアモジュールの動作状態を監視するためのレジスタであり、各ハードウェアモジュールに対応したビットを持たせている。このレジスタの値は、ＣＰＵ１１がセットし、各ハードウェアモジュールがリセットする。このレジスタは、従来のシステムＬＳＩのＣＰＵ１１が同等の機能を有するレジスタを持つ場合は、それで代用することができる。ウェイクアップモジュールレジスタ２２は、スリープ命令コード中の、ウェイクアップモジュールフィールドＷＵの値を格納するレジスタであり、ＣＰＵ１１がスリープ命令をデコード／実行するときに格納値の転送を行う。コンパレータ２１は、ＣＰＵ１１をウェイクアップさせるタイミングを検知するための回路である。モジュール状態フラグレジスタ２３の値、ウェイクアップモジュールレ

ジスタ２２の値、スリープ命令コード中の条件フィールドＣＯの値、及び割込み信号ＢＲが入力され、それらの値によって、スリープ制御フリップフロップ２４に対してリセット信号ＲＴＦを出力する。

【００２８】本実施形態のシステムＬＳＩの動作について説明する。

【００２９】初期状態、すなわちシステムＬＳＩがリセットされたときには、コンパレータ２１に割込み信号ＢＲ（リセット割込み）が入力される。これによって、スリープ制御フリップフロップ２４がリセットされて、スリープ制御フリップフロップ２４の出力ＯＵＴが“０”になるため、ＣＰＵ１１にクロックＣＬＫが供給される。ＣＰＵ１１は、あるハードウェアモジュールに処理命令を出すと同時に、モジュール状態フラグレジスタ２３中の、そのハードウェアモジュールに対応するビットをセットする。

【００３０】次にスリープ命令を実行すると、スリープ命令コード中のウェイクアップモジュールフィールドＷＵの値と条件フィールドＣＯの値が、それぞれパワーマネジメントモジュール２０のウェイクアップモジュールレジスタ２２とコンパレータ２１に転送される。また、スリープ制御フリップフロップ２４がセットされ、その出力ＯＵＴが“１”となる。

【００３１】ＯＲゲート２５の入力の１つであるスリープ制御フリップフロップ２４の出力ＯＵＴが“１”となると、ＯＲゲート２５の出力、すなわち、ＣＰＵ１１へのクロック信号ＣＬＫが、もう一方の入力であるメインクロックＣＫの値に関わらず、“１”となる。つまり、ＣＰＵ１１へのクロックＣＬＫの供給が停止されたことになるので、ＣＰＵ１１はスリープ状態となる。

【００３２】各ハードウェアモジュールＭ０～Ｍ４は、与えられた処理が終了すると、モジュール状態フラグレジスタ２３の、そのハードウェアモジュールに対応したビットをリセットする。コンパレータ２１は、モジュール状態フラグレジスタ２３とウェイクアップモジュールレジスタ２２の値とを常に比較しており、その結果が、スリープ命令コードの条件フィールドＣＯで与えられた条件を満足した場合に、スリープ制御フリップフロップ２４に対してリセット信号ＲＴＦを出力する。また、割込み信号ＢＲが入力された場合は、比較結果に関わらずリセット信号ＲＴＦを出力する。

【００３３】スリープ制御フリップフロップ２４がリセットされると、このフリップフロップ２４の出力ＯＵＴが“０”となるため、ＯＲゲート２５のもう一つの入力であるメインクロックＣＫが、ＯＲゲート２５の出力ＣＬＫにそのまま伝達される。これによって、ＣＰＵ１１へのクロックＣＬＫの供給が再開され、ＣＰＵ１１がウェイクアップする。

【００３４】図６に示すような、３つのハードウェアモジュールＭ０，Ｍ１，Ｍ２を搭載したシステムＬＳＩを

例にして、上述の動作をさらに具体的に説明する。

【００３５】図７は、本例におけるパワーマネジメントモジュール２０内のコンパレータ２１の構成例を示す回路図である。ここで、ハードウェアモジュールＭ０，Ｍ１，Ｍ２に対応するモジュール状態フラグレジスタ２３及びウェイクアップモジュールレジスタ２２の各ビットを、それぞれ（Ｓ０，Ｓ１，Ｓ２）及び（Ｗ０，Ｗ１，Ｗ２）とする。（Ｓ０，Ｓ１，Ｓ２）は値が“１”のときに、それらに対応するハードウェアモジュールが処理実行中であることを示し、（Ｗ０，Ｗ１，Ｗ２）は値が“１”のときに、それらに対応するハードウェアモジュールがウェイクアップモジュールとして指定されたことを示すものとする。

【００３６】ＣＰＵ１１からハードウェアモジュールＭ１に対して処理命令が出されると、（Ｓ０，Ｓ１，Ｓ２）＝（１，０，０）となる。また、ハードウェアモジュールＭ０が処理を実行しているときに、ハードウェアモジュールＭ２にも処理命令が出されると、（Ｓ０，Ｓ１，Ｓ２）＝（１，０，１）となる。その後、ハードウェアモジュールＭ０とＭ２の両方の処理終了までＣＰＵ１１をスリープさせる図８のスリープ命令がデコード／実行されると、まず、（Ｗ０，Ｗ１，Ｗ２）に（１，０，１）が、コンパレータ２１にＡＮＤ条件を表す“０”が、それぞれ転送される。

【００３７】次に、スリープ制御フリップフロップ２４にセット信号ＳＴＦが入力されることによって、ＣＰＵ１１がスリープ状態となる。コンパレータ２１内では、（Ｓ０，Ｓ１，Ｓ２）と（Ｗ０，Ｗ１，Ｗ２）の値が組合せ回路（ＡＮＤ条件ブロック２１ａ、ＯＲ条件ブロック２１ｂ）によって比較されている。スリープ命令によって、ＡＮＤ条件が指定されているので、マルチプレクサ２１ｃによってＡＮＤ条件ブロック２１ａの出力が選択される。

【００３８】ＡＮＤ条件ブロック２１ａでは、モジュール状態フラグレジスタ２３とウェイクアップモジュールレジスタ２２のハードウェアモジュールＭ０，Ｍ１，Ｍ２に対応するビットが、それぞれＮＡＮＤゲートＡＣ０，ＡＣ１，ＡＣ２に入力される。（Ｓ０，Ｓ１，Ｓ２）＝（１，０，１），（Ｗ０，Ｗ１，Ｗ２）＝（１，０，１）のときは、ＮＡＮＤゲートＡＣ０，ＡＣ１，ＡＣ２の出力は、それぞれ“０”，“１”，“０”であるから、これらの値を入力とするＡＮＤゲートＡＣ３の出力は“０”となる。Ｗ１＝０であるから、Ｓ１の値つまりハードウェアモジュールＭ１の動作状態に関わらず、ＮＡＮＤゲートＡＣ１の出力は常に“１”となる。すなわち、ウェイクアップモジュールとして指定されたハードウェアモジュールの動作状態だけがＡＮＤゲートＡＣ３の出力に影響する。

【００３９】ハードウェアモジュールＭ０の処理が終了し、Ｓ０＝０となって、ＮＡＮＤゲートＡＣ０の出力が

"１"となっても、ＮＡＮＤゲートＡＣ２の出力が"０"であるから、ＡＮＤゲートＡＣ３の出力は"０"のままである。ハードウェアモジュールＭ２の処理も終了し．Ｓ２＝０となると．ＮＡＮＤゲートＡＣ２の出力が"１"となる。ＮＡＮＤゲートＡＣ０，ＡＣ１．ＡＣ２全ての出力が"１"となると、ＡＮＤゲートＡＣ３の出力が"１"となる。ＡＮＤゲートＡＣ３の出力がマルチプレクサ２１ｃによって選択されているので．マルチプレクサ２１ｃの出力と．割込み信号ＢＲがＯＲゲート２１ｄの入力となっているので、割込み信号ＢＲが入力されていないときは、ＡＮＤゲートＡＣ３の出力がそのままＯＲゲート２１ｄの出力に伝送される。これがコンパレータ２１の出力、すなわちリセット信号ＲＴＦとなるので、ハードウェアモジュールＭ０とＭ２の両方の処理が終了したときに、リセット信号ＲＴＦが"１"となる。このリセット信号ＲＴＦによって、スリープ制御フリップフロップ２４がリセットされ．その出力ＯＵＴが"０"となると、メインクロックＣＫの値が．ＯＲゲート２５の出力にそのまま伝送され．ＣＰＵ１１へのクロックＣＬＫの供給が再開される。

【００４０】次に、ハードウェアモジュールＭ０あるいはＭ２の処理の終了までＣＰＵ１１をスリープさせる図９のスリープ命令が実行された場合について説明する。

【００４１】スリープ命令によって．ＯＲ条件が指定されているので．マルチプレクサ２１ｃによって．ＯＲ条件ブロック２１ｂの出力が選択される。ＯＲ条件ブロック２１ｂでは．モジュール状態フラグレジスタ２３とウェイクアップモジュールレジスタ２２のＭ０，Ｍ１，Ｍ２に対応するビットが、それぞれＡＮＤゲートＯＣ０，ＯＣ１，ＯＣ２に入力される。ただし．モジュール状態フラグレジスタ２３の値は．反転された値が入力される。（Ｓ０，Ｓ１，Ｓ２）＝（１，０．１），（Ｗ０，Ｗ１．Ｗ２）＝（１，０．１）のときは、ＡＮＤゲートＯＣ０，ＯＣ１．ＯＣ２の出力は、それぞれ"０"，"０"，"０"であるから．これらの値を入力とするＯＲゲートＯＣ３の出力は"０"となる。Ｗ１＝０であるから．Ｓ１の値つまりハードウェアモジュールＭ１の動作状態に関わらず、ＡＮＤゲートＯＣ１の出力は"０"常にとなる。すなわち、ウェイクアップモジュールとして指定されたハードウェアモジュールの動作状態だけが．ＯＲゲートＯＣ３の出力に影響する。

【００４２】ハードウェアモジュールＭ０の処理が終了し．Ｓ０＝０となると、ＡＮＤゲートＯＣ０の出力が"１"となり．これによって、ＯＲゲートＯＣ３の出力も"１"となる。この値がマルチプレクサ２１ｃとＯＲゲート２１ｄを通して、リセット信号ＲＴＦとしてスリープ制御フリップフロップ２４に出力される。

【００４３】割込み信号ＢＲが入力された場合は．マルチプレクサ２１ｃの出力に関係なくＯＲゲート２１ｄの出力が"１"となり、スリープ制御フリップフロップ２４に対して、リセット信号ＲＴＦが出力されることになる。

【００４４】ここでは、ＣＰＵ１１をウェイクアップさせる条件を、ＡＮＤ条件とＯＲ条件のみで構成と動作を説明したが、条件ブロックの回路を追加し、スリープ命令コードの条件フィールドＣＯに複数ビットを使用することで、他の条件論理も実現できる。

【００４５】また、特定のハードウェアモジュールの処理終了によって．ＣＰＵ１１をウェイクアップさせる、あるいはどのハードウェアモジュールの処理が終了してもＣＰＵ１１をウェイクアップさせる場合には．コンパレータ２１、ウェイクアップモジュールレジスタ２２、及びモジュール状態フラグレジスタ２３は必要なくなる。ハードウェアモジュールからの処理終了信号と割込み信号ＢＲのＯＲ演算の結果をリセット信号ＲＴＦとしてスリープ制御フリップフロップ２４に入力すればよい。これによって、パワーマネジメントモジュール２０を簡略化することができる。

【００４６】割込み信号ＢＲが入力されると、スリープ制御フリップフロップ２４に対してリセット信号ＲＴＦが出力されることを利用して、回路がある状態になった時にＣＰＵ１１をウェイクアップさせることが可能となる。例えばデータバスが所定の状態になったときにＣＰＵ１１をウェイクアップさせるのであれば、データバスの状態を監視する回路を付加し、データバスがその状態になったときに．割込み信号ＢＲを出力するようにすればよい。

【００４７】本実施形態では、従来のシステムＬＳＩにパワーマネジメントモジュール２０を追加した構成で説明をしたが、これは説明を容易にするためであり．この機能をＣＰＵ１１の中に含めてもよい。この場合は、ウェイクアップモジュールレジスタ２２．モジュール状態フラグレジスタ２３及びスリープ制御フリップフロップ２４に相当する回路には．常にクロックが供給されるようにすることで．同様の動作が実現できる。

【００４８】このように本実施形態では、ソフトウェア（スリープ命令）によってＣＰＵ１１をスリープさせるタイミングを制御するようにしたので．プログラマがＣＰＵ１１をいつスリープさせるかを決定することができ．無駄なスリープの実行による性能低下を避けることが可能となる。さらに、ハードウェアによってＣＰＵ１１のウェイクアップを制御することによって、スリープ期間を予測すること無しに適切な期間だけスリープさせることが可能となる。これにより、無駄なスリープの実行による性能低下を避けつつ低消費電力化効果を大幅に向上させることが可能となる。また．これらの機能を実現するパワーマネジメントモジュール２０は、非常に簡単なハードウェアで実装することができるため．これによる面積、消費電力のオーバヘッドも小さい。

【００４９】複数の命令で構成されていたハードウェアモジュール処理終了待ちルーチンを、１つのスリープ命令に置き換えるので、プログラムを格納するために必要となるメモリ量が小さくなり、また、実行される命令数も従来のプログラムより少なくなる。さらに、ＣＰＵがスリープしている間は、命令メモリはアクセスされないので、この間は、命令メモリもスリープさせることができる。これらのことによって、命令メモリの消費電力も削減される。

【００５０】［第２実施形態］本発明の第２実施形態を図１０、図１１及び図１２を参照して説明する。

【００５１】５つのハードウェアモジュールＭ０、Ｍ１、Ｍ２，Ｍ３、Ｍ４を持つ図１に示したシステムＬＳＩにおいて、実行されるプログラムの一部が図１０に示すようなフローチャートで表される場合を考える。ここで、処理１．１、処理１．２、処理２．１、処理２．２、及び処理３で、それぞれハードウェアモジュールＭ０、Ｍ１，Ｍ２、Ｍ３，Ｍ４に対して処理命令が出されるとする。処理３が終了した後に、ハードウェアモジュール処理終了待ちルーチンがあるため、これをスリープ命令に置き換えることによって、ＣＰＵ１１をスリープさせ、消費電力の削減を図る。

【００５２】しかし、このルーチンが、実行された処理において処理命令を与えられた全てのハードウェアモジュールの処理が終了するのを待つものであるとすると、図２のスリープ命令コードでは実現できない。なぜなら、図２のスリープ命令では、ウェイクアップ命令を明示的に指定する必要があるのに対し、このプログラムでは、条件分岐があるために、処理命令を出されるハードウェアモジュールが１組に決定されないからである。

【００５３】そこで、図１１に示す第２のスリープ命令コードを導入する。このスリープ命令コードは、オペコードＯＰと１つのレジスタフィールドＲＧとからなる。レジスタフィールドＲＧでは、あるレジスタを１つ指定する。ここで指定されるレジスタには、図２のスリープ命令コードでのウェイクアップモジュールフィールドＷＵと条件フィールドＣＯの値に相当する内容を格納しておく。このスリープ命令をデコードし実行するときに、レジスタフィールドＲＧで指示されたレジスタの値をパワーマネジメントモジュール２０に転送することによって、ＣＰＵ１１のウェイクアップを制御する。

【００５４】次に、このスリープ命令を用いて、ＣＰＵ１１をスリープ状態にする方法を図１２のフローチャートを参照して説明する。

【００５５】レジスタフィールドＲＧで指定されたレジスタの最上位ビットが条件フィールドＣＯに相当し、最下位ビットから５ビットがウェイクアップモジュールフィールドＷＵに相当するとする。また、スリープ命令では、汎用レジスタｒ８が指定されるとする。

【００５６】図１２において、まず、スリープ動作に関わる処理を行う前に、「ｒ８」に「０ｘ０００」を代入することによって初期化しておく。処理１．１においてハードウェアモジュールＭ０に処理命令を出した後に、「ｒ８」の値と「０ｘ０００１」をＯＲ演算を行う。同様に、処理１．２、処理２．１、処理２．２、及び処理３において、ハードウェアモジュールＭ１、Ｍ２、Ｍ３，Ｍ４に処理命令を出した後に、「ｒ８」の値と「０ｘ０００２」、「０ｘ０００４」、「０ｘ０００８」、「０ｘ００１０」とのＯＲ演算をそれぞれ行い、その結果を「ｒ８」に格納しておく。条件フィールドＣＯの値を指定するために、スリープ命令を実行する前に、ＡＮＤ条件ならば「０ｘ０００」、ＯＲ条件なら「０ｘ８０００」と「ｒ８」とのＯＲ演算をそれぞれ行う。

【００５７】このようにすることによって、例えば、処理１．１、処理２．１を行い、ＡＮＤ条件でＣＰＵ１１をウェイクアップさせる場合には、「ｒ８」の内容が「０ｘ００１５」となり、ハードウェアモジュールＭ０とＭ２のＡＮＤ条件によって、ＣＰＵ１１をウェイクアップさせるというスリープ動作を実現することができる。

【００５８】なお、パイプライン処理をしているために、スリープ命令が実行段に入る前に先行する命令を完了させたい場合がある。例えば、ロード／ストア命令など、ＣＰＵ１１での処理でメモリ１２にアクセスする場合、ＣＰＵ１１からのアクセス要求が出たままになると、他モジュールがアクセスできなくなる場合である。その場合には、スリープ命令の前に必要な数のＮＯＰ命令を挿入し、その命令が終了してからスリープ命令が実行されるようにする。

【００５９】次に、上記第１及び第２実施形態に用いられる、ＣＰＵ１１に対するスリープ及びウェイクアップ処理の実行手順を、図１３のフローチャートを参照して説明する。

【００６０】まず、ＣＰＵ１１が次に実行する命令をデコードし（ステップＳ１１）、それが、ＣＰＵ１１をスリープさせるスリープ命令であるかを判断する（ステップＳ１２）。スリープ命令であれば、この命令によって指示される、ＣＰＵ１１をウェイクアップさせるための条件がレジスタに格納される（ステップＳ１３）。さらに、ＣＰＵ１１へのクロックの供給を停止させる信号が生成され、この信号によってＣＰＵ１１がスリープする（ステップＳ１４）。

【００６１】ＣＰＵ１１がスリープしている時に、あるハードウェアモジュールの実行中の処理が終了すると、処理終了信号が生成される（ステップＳ１５）。そのハードウェアモジュールの処理終了によって、格納しておいたＣＰＵ１１へのクロック供給を再開させる条件が満たされたか否かが判断され（ステップＳ１６）、満たされていれば、ＣＰＵ１１へのクロックの供給を再開させ

る信号が生成され、ＣＰＵ１１がウェイクアップする（ステップＳ１７）。

【００６２】〔第３実施形態〕上記第１及び第２実施形態では、あるハードウェアモジュールの処理が終了するまでＣＰＵ１１が次の処理を実行できない場合において、ＣＰＵ１１をソフトウェア（スリープ命令）によってスリープ状態にし、そのハードウェアモジュールからの処理終了信号によってＣＰＵ１１をウェイクアップさせている。この際に用いられるスリープ命令コードは、スリープ命令を表すオペコードＯＰと、どのハードウェアモジュールからの処理終了信号によってＣＰＵ１１をウェイクアップさせるかを指定するウェイクアップモジュールフィールドＷＵと、ウェイクアップモジュールフィールドＷＵで複数のハードウェアモジュールが指定された時にそれらの全て処理が終了したらＣＰＵ１１をウェイクアップさせるＡＮＤ条件、及びそれらのいずれかの処理が終了したらＣＰＵ１１をウェイクアップさせるＯＲ条件を表す条件フィールドＣＯとからなる。上記第１及び第２実施形態では、このようなスリープ命令コードをコンパイラにより生成することができないため、人手によってハードウェアモジュール処理終了待ちルーチンをスリープ命令に置き換えなければならないという不具合があった。

【００６３】この不具合を解消するために、本第３実施形態では、従来のプログラムにおけるハードウェアモジュール処理終了待ちルーチンを前記スリープ命令に自動的に置き換える機能を設けている。以下、その手法について詳細に説明する。

【００６４】図１４は、システムＬＳＩのプログラムにおけるハードウェアモジュール処理終了待ちルーチンの一例を示す図である。

【００６５】同図において、まず、ｍｆｃ命令によって、ハードウェアモジュールの状態レジスタｓｔａｔｅの値を汎用レジスタｒ２にロードする。ここで、状態レジスタｓｔａｔｅの各ビットは、対応するハードウェアモジュールが処理を実行中であれば"１"、処理を終了していれば"０"とする。

【００６６】続いてａｎｄｉ命令によってロードした値にマスクをかける。これは、ａｎｄｉ命令の第３オペランドが「０ｘ０００３」であるから、状態レジスタｓｔａｔｅの第０ビット（ＬＳＢ）と第１ビットに対応するハードウェアモジュール（以下、それぞれＭ０、Ｍ１とする）の動作状態のみを取り出している。そして、ｂｎｅ命令によって、常に"０"であるレジスタｒ０との比較を行い、その結果が等しくなければ、もう一度、ｍｆｃ２命令に分岐し、等しければ、そのルーチンを抜ける。つまり、このルーチンは、ハードウェアモジュールＭ０、Ｍ１の両方の処理が終了するのを待つルーチンということになる。

【００６７】次に、このルーチンをスリープ命令に置き換えることを考える。

【００６８】本例の場合は、ハードウェアモジュールＭ０、Ｍ１の処理の終了によって、ＣＰＵ１１をウェイクアップさせるのであるから、スリープ命令のウェイクアップモジュールフィールドＷＵは、「０ｘ０００３」となる。また、ハードウェアモジュールＭ０とＭ１の両方、つまりウェイクアップモジュールフィールドＷＵで指定した全てのハードウェアモジュールの処理が終了したら、ＣＰＵ１１をウェイクアップさせるので、条件フィールドＣＯの値はＡＮＤ条件となる。したがって、対応するスリープ命令は、図１５に示すようになる。ここで、スリープ命令を表すオペコードＯＰを"１１１１１"、条件フィールドＣＯの値は、ＡＮＤ条件が"０"、ＯＲ条件が"１"とする。

【００６９】ハードウェアモジュール処理終了待ちルーチンのもう一つの例を、図１６に示す。この例では、まず、レジスタｒ３に「０ｘ０００３」を代入しておき、これをｍｆｃ２命令の次のａｎｄ命令でマスクとして用いている。マスクをかけた状態レジスタｓｔａｔｅの値がｒ３と同じであった場合、つまりハードウェアモジュールＭ０，Ｍ１のいずれの処理も終了していない場合は、分岐が成立し、異なっていた場合、つまり、ハードウェアモジュールＭ０またはＭ１のいずれかの処理が終了していた場合に、このルーチンから抜ける。すなわち、このルーチンは、ハードウェアモジュールＭ０またはＭ１のいずれかの処理が終了するのを待つためのルーチンである。したがって、このルーチンをスリープ命令に置き換えると、図１７に示すようになる。ここで、ウェイクアップモジュールフィールドＷＵの値は、ハードウェアモジュールＭ０及びＭ１を示す「０ｘ０００３」であり、条件フィールドＣＯは、ハードウェアモジュールＭ０またはＭ１のいずれかの処理が終了するまでＣＰＵ１１をスリープさせるので、ＯＲ条件となる。

【００７０】コンパイラにおいて、このような命令系列を生成する手続きを、スリープ命令を生成する手続きに置き換えることによって、スリープ命令を自動的に生成することができる。

【００７１】次に、ハードウェアモジュール処理終了待ちルーチンをスリープ命令に置き換えるための手続きについて、図１８及び図１９を参照して説明する。

【００７２】図１８のフローチャートにおいて、まず、ソースコードをコンパイラによって、アセンブリコードに変換する（ステップＳ２１）。次に、これをフィルタリングすることによって、ハードウェアモジュール処理終了待ちルーチンをスリープ命令に変換する（ステップＳ２２）。最後に、変換したアセンブリコードをアセンブルすることによって、スリープ命令を含む機械語を生成する（ステップＳ２３）。

【００７３】ステップＳ２２でのフィルタリングには、変換テーブルを用いる。変換テーブルは、ハードウェア

モジュール処理終了待ちルーチンを実現している命令系列と．その命令系列に対応したスリープ命令の組み合わせとからなる。命令系列のオペランドは、定数と変数によって表現する。図１９に．図１４と図１５の命令系列をスリープ命令に変換するための変換テーブルの例を示す。

【００７４】図１９において、まず．変数と定数の宣言を行う。ここでは、＄ｃｏｎｓｔ以下のｓｔａｔｅ、ｒ０．ａｎｄ、ｏｒが定数であり、＄ｖａｒ以下のｒｅｇ１．ｒｅｇ２．ｉｍｍ１．ｉｍｍ２が変数である。＄ｌａｂｅｌ以下のＬＡＢＥＬ１とＬＡＢＥ２は、分岐先が一致するかの判定のための特殊変数（以下、ラベル変数と呼ぶ）である。＄ｔａｂｌｅと＄ｅｎｄｔａｂｌｅで囲まれた部分が変換テーブルの実態であり、この中の＄ｓｔｒｅａｍ以下が命令系列、＄ｆｏｒｍａｔ以下が変換するスリープ命令のフォーマットである。

【００７５】さらに、フィルタリングの手続きについて説明する。フィルタは、アセンブリコードの最初から順に．変換テーブルの命令系列との比較を行っていく。比較は次のように行う。まず．オペコードＯＰが一致するかを判定する。一致したら．次にオペランドが一致するかを判定する。この際に．変換テーブルで定数として与えられているオペランドは．そのまま比較を行う。変数として与えられているオペランドには．その変数を用いて行う最初の比較の際に．アセンブリコードのオペランドの値を代入し．次にその変数との比較を行う時には、代入した値を用いて比較を行う。比較の結果、両者が一致したら、アセンブリーコード中の命令系列を．＄ｆｏｒｍａｔで与えられた、スリープ命令のフォーマットに従って置き換える。

【００７６】以下、あるアセンブリコード中に図１４に示すようなハードウェアモジュール処理待ちルーチンがある場合に、これを図１９に示した変換テーブルでスリープ命令に変換する際の手順について具体的に説明する。

【００７７】アセンブリコード中に「ＬＡＢＥＬ１：」が現れたので．このラベルの位置を格納しておく。次に．ｍｆｃ２命令があり．これが、変換テーブル中の命令系列の最初の命令と一致するので．そのオペランドの比較を行う。変換テーブル中のｍｆｃ２命令の第１オペランドがｒｅｇ１という変数であるから、これに．アセンブリコード中のｍｆｃ２命令の第１オペランドの値ｒ２が代入される。変換テーブルのｍｆｃ２命令の第２オペランドは定数であるから．そのまま比較を行う。

【００７８】ｍｆｃ命令はオペランドも一致するので、次の命令の比較を行う。両者ともａｎｄｉ命令で一致するので、第１オペランドの比較を行う。変換テーブルのａｎｄｉ命令の第１オペランドは変数ｒｅｇ２なので、これにアセンブリコード中のａｎｄｉ命令の第１オペランドｒ２が代入される。

【００７９】変換テーブルのａｎｄｉ命令の第２オペランドは既出のｒｅｇ１であるから、先にこの変数に代入したｒ２を用いて比較する。第３オペランドの比較では．ｉｍｍ１に「０ｘ０００３」が代入される。次のｂｎｅ命令の比較では、第２オペランドまでは同様である。

【００８０】変換テーブルのｂｎｅ命令の第３オペランドのＬＡＢＥＬ１は、ラベル変数であるから、アセンブリコード中のｂｎｅ命令の第３オペランドのラベルと分岐先が同じであるかの判定を行う。この例の場合．双方ともｍｆｃ２命令に分岐しているので．一致する。変換テーブルの命令系列と全て一致したので、アセンブリコード中のその命令系列を．＄ｆｏｒｍａｔで与えられたフォーマットに従ってスリープ命令に置き換える。この例の場合、ｉｍｍ１が変数であるから．ａｎｄｉ命令の比較の時に代入された「０ｘ０００３」が用いられる。

【００８１】このように．本実施形態では、従来のプログラム中のハードウェアモジュール処理終了待ちルーチンであった命令系列を自動的にスリープ命令に置き換えることができるので、その分、プログラマの労力が減少し．システムＬＳＩにおける低消費電力設計の効率化を図ることが可能になる。

【００８２】

【発明の効果】以上詳細に説明したように、請求項１乃至請求項４記載の発明であるシステムＬＳＩによれば、ソフトウェア（スリープ命令）によってＣＰＵに対するクロック供給の停止（スリープ）を制御することができるので、無駄なスリープの実行による性能低下を避けつつ低消費電力化を図ることが可能となる。さらに．ハードウェアによってＣＰＵに対するクロック供給の再開（ウェイクアップ）を制御することによって、スリープ期間を予測すること無しに適切な期間だけスリープさせることができ．無駄なスリープの実行による性能低下を避けつつ低消費電力化効果を大幅に向上させることが可能となる。また．これらの機能を実現するパワーマネジメント装置は．非常に簡単なハードウェアで実装することができるため．これによる面積、消費電力のオーバヘッドも小さい。また、従来のプログラム中に存在するハードウェアモジュール処理終了待ちルーチンを１つのスリープ命令に置き換えることにより．命令の実行数が従来のプログラムより少なくなるため．メモリアクセスによる消費電力も削減することができる。さらに．複数の命令を１命令に置き換えるので、プログラムを格納するために必要となるメモリ量も小さくなる。

【００８３】請求項５記載の発明であるシステムＬＳＩによれば、スリープ命令は．第１のフィールドの値を格納したレジスタを指定する第２のフィールドを有し、パワーマネジメント装置は．第１と第２のフィールドの内容を参照してＣＰＵに対するクロック供給を再開する構成にしたので．例えば条件分岐を含むプログラムにおい

て．処理命令を出されるハードウェアモジュールが１組に決定されないような場合であっても．ＣＰＵに対するクロック供給の再開を的確に制御することができ．上記発明と同等の効果を享受することが可能になる。

【００８４】請求項６記載の発明であるシステムＬＳＩによれば、ハードウェアモジュールの処理の終了を待つために用いられるプログラム中のルーチンをスリープ命令に自動的に置き換えるようにしたので、システムＬＳＩにおける低消費電力設計の効率化を図ることが可能になる。

【００８５】請求項７記載の発明であるパワーマネジメント方法によれば、上記請求項１乃至請求項４記載の発明と同等の効果を享受することができる。

【００８６】また、複数の命令で構成されていたハードウェアモジュール処理終了待ちルーチンを、１つのスリープ命令に置き換えるので．プログラムを格納するために必要となるメモリ量が小さくなり．また、実行される命令数も従来のプログラムより少なくなる。さらに、ＣＰＵがスリープしている間は、命令メモリはアクセスされないので、この間は、命令メモリもスリープさせることができる。これらのことによって．命令メモリの消費電力も削減される。

【図面の簡単な説明】

【図１】本発明の第１実施形態に係るシステムＬＳＩの構成を示すブロック図である。

【図２】スリープ命令コードの構成を示す図である。

【図３】スリープ命令コードの具体的構成例を示す図である。

【図４】システムＬＳＩのプログラム中のルーチンを示す図である。

【図５】パワーマネジメントモジュール２０の構成を示すブロック図である。

【図６】第１実施形態に係るシステムＬＳＩの具体的構成例を示すブロック図である。

【図７】図６に示したシステムＬＳＩの構成においてパワーマネジメントモジュール内のコンパレータの構成例を示す回路図である。

【図８】図６に示したシステムＬＳＩに使用されるスリープ命令コードの具体的構成例を示す図である。

【図９】図６に示したシステムＬＳＩに使用されるスリ＊40＊ープ命令コードの他の具体的構成例を示す図である。

【図１０】本発明の第２実施形態を説明するためのプログラムのフローチャートである。

【図１１】第２実施形態に係るスリープ命令コードの構成を示す図である。

【図１２】第２実施形態に係るＣＰＵのスリープ動作を説明するフローチャートである。

【図１３】ＣＰＵ１１に対するスリープ及びウェイクアップ処理の実行手順を示すフローチャートである。

【図１４】システムＬＳＩのプログラムにおけるハードウェアモジュール処理終了待ちルーチンの一例を示す図である。

【図１５】図１４に示した例に係る．第３実施形態のスリープ命令コードの構成を示す図である。

【図１６】システムＬＳＩのプログラムにおけるハードウェアモジュール処理終了待ちルーチンの他の例を示す図である。

【図１７】図１６に示した例に係る．第３実施形態のスリープ命令コードの構成を示す図である。

【図１８】ハードウェアモジュール処理終了待ちルーチンをスリープ命令に置き換えるための第３実施形態の処理を示すフローチャートである。

【図１９】図１４と図１５の命令系列をスリープ命令に変換するための変換テーブルの例を示す図である。

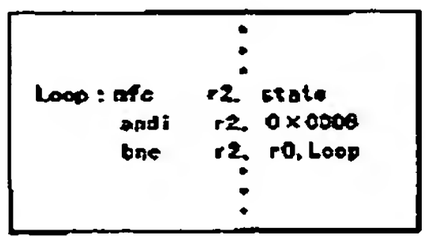
【図２０】従来のシステムＬＳＩの構成を示すブロック図である。

【符号の説明】

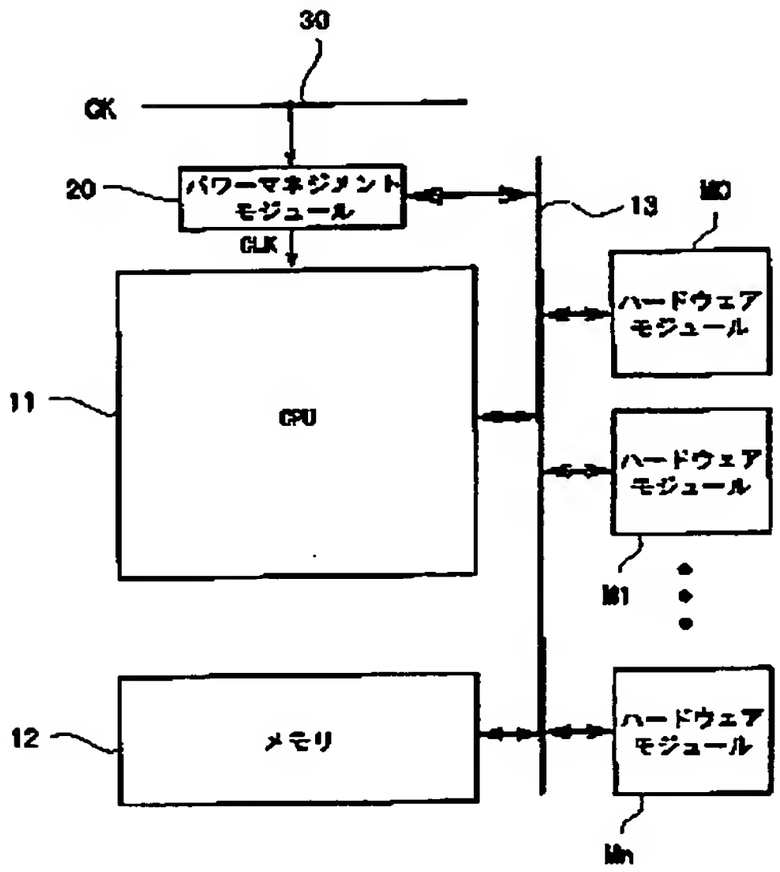
１１　ＣＰＵ
１２　メモリ
２０　パワーマネジメントモジュール
２１　コンパレータ
２２　ウェイクアップモジュールレジスタ
２３　モジュール状態フラグレジスタ
２４　スリープ制御フリップフロップ
２５　ＯＲゲート
ＯＰ　スリープ命令を表すオペコード
ＷＵ　ウェイクアップモジュールフィールド
ＣＯ　条件フィールド
ＲＧ　レジスタフィールド
Ｍ１～Ｍｎ　ハードウェアモジュール

【図２】

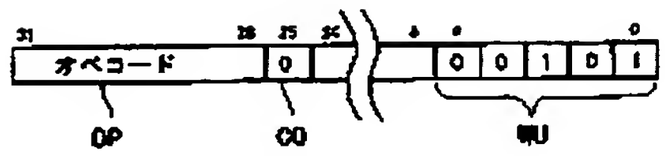
【図４】



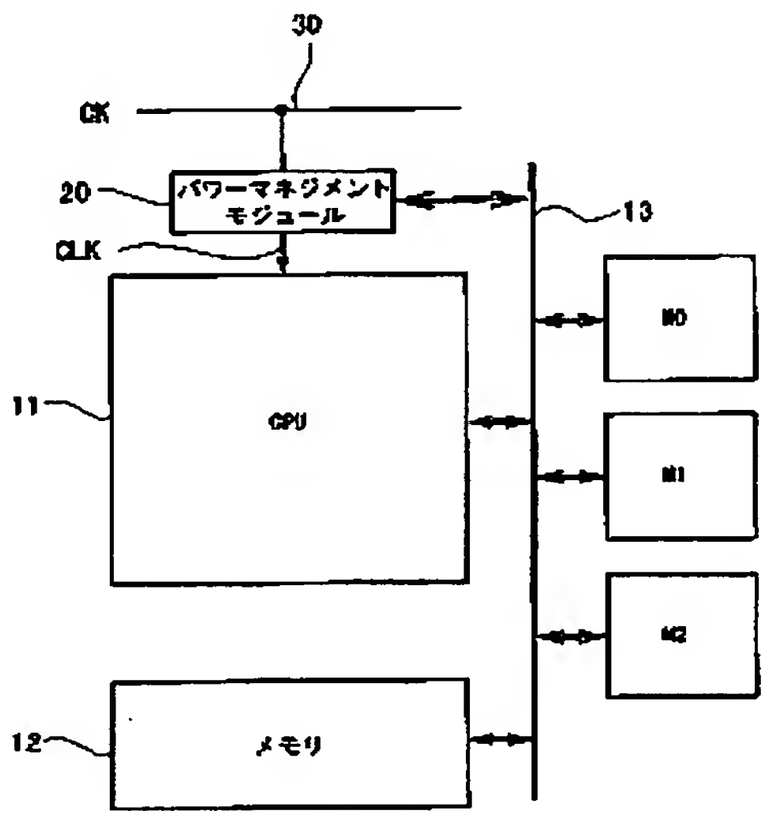
【図１】



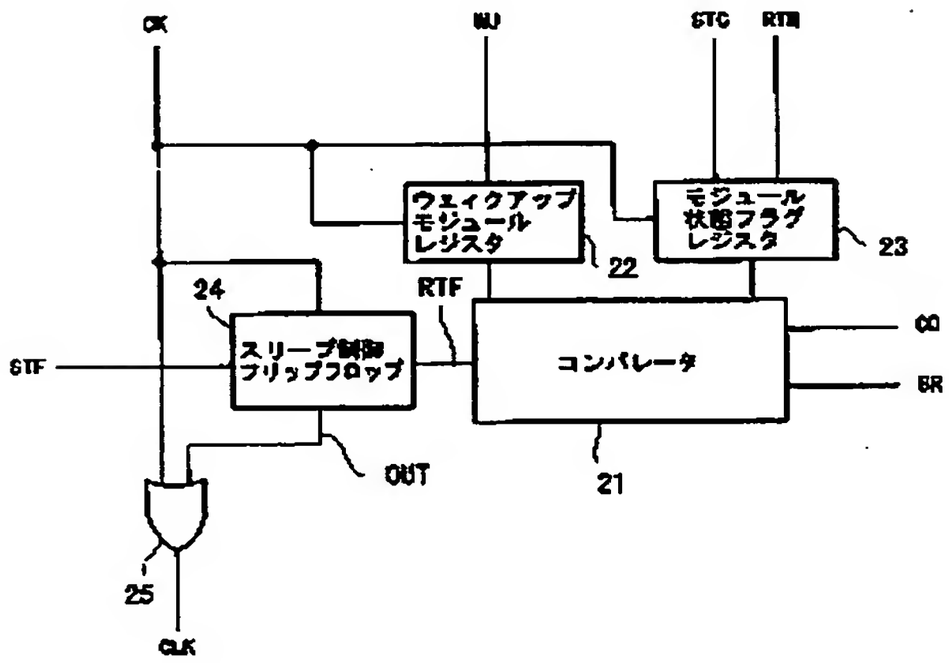
【図３】



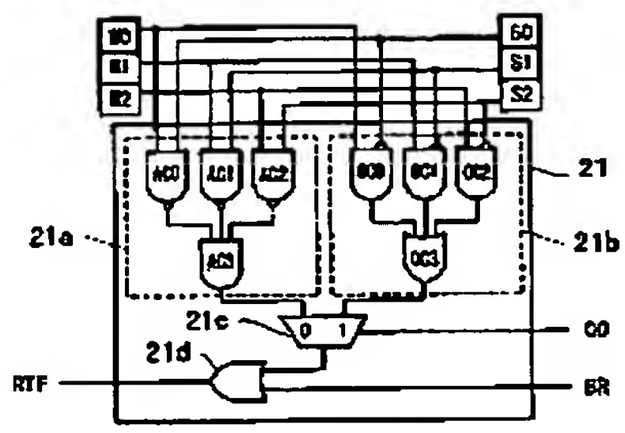
【図６】



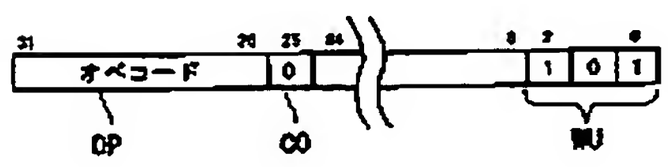
【図５】



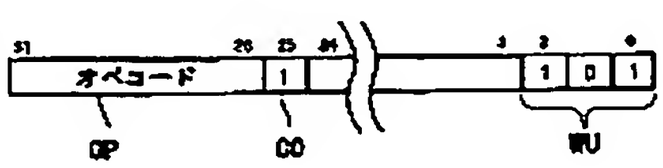
【図７】



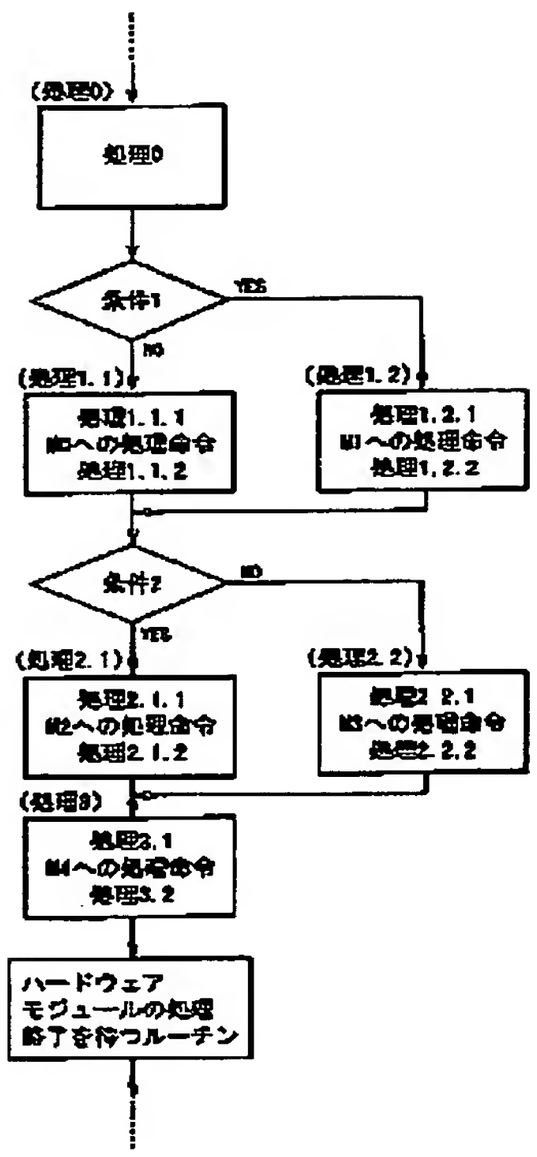
【図８】



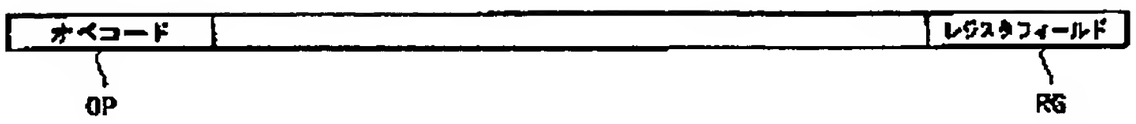
【図９】



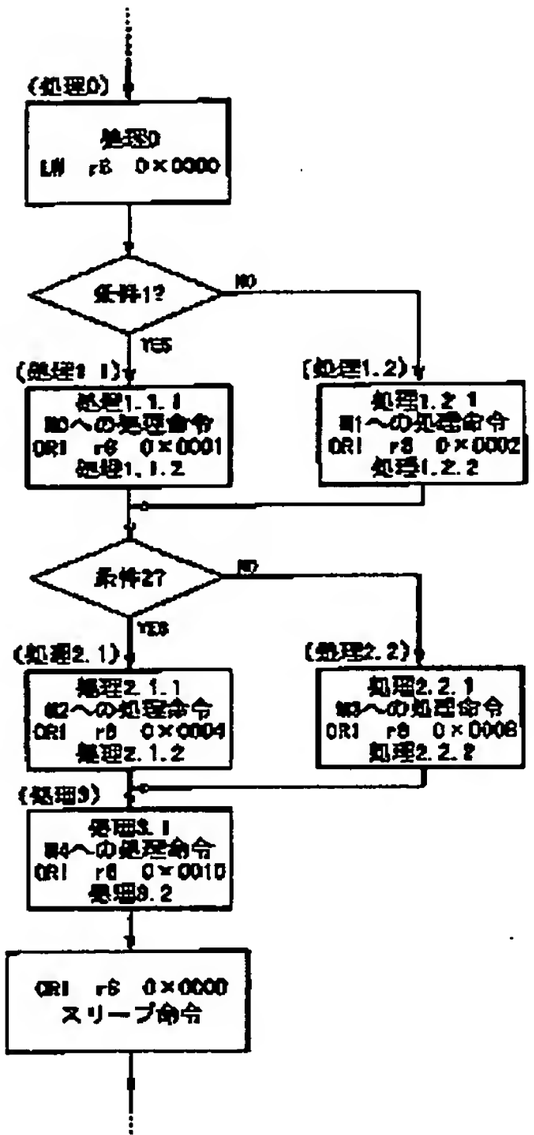
【図１０】



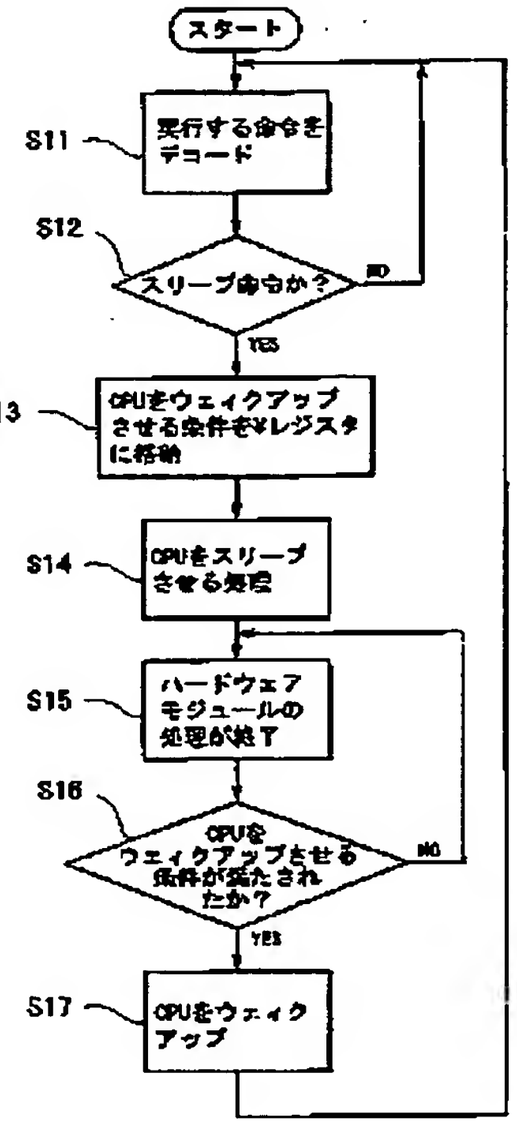
【図１１】



【図１２】



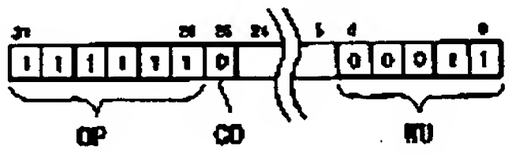
【図１３】



【図１４】

```
LABEL1 : mfc   r2, state
         andi  r2, r2, 0×0003
         bne   r2, r0, LABEL1
         nop
```

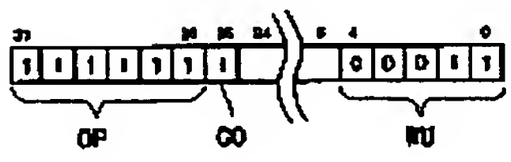
【図１５】



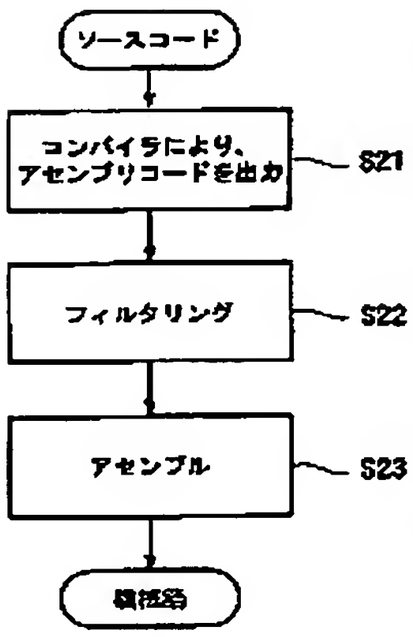
【図１６】

```
        add   r3, r0, 0×0003
LABEL2 : mfc  r2, state
         and  r2, r2, r3
         beq  r2, r3, LABEL2
         nop
```

【図１７】



【図１８】



【図１９】

```
$const
  state  r0  and  or
$var
  reg1  reg2  reg3  reg4  imm1  imm2
$label
  LABEL1  LABEL2
$table
$stream
  LABEL1:mfc   reg1, state
               andi  reg2, reg1, imm1
               bne   reg2, r0, LABEL1
               nop
$format
               sleep  and, imm1
$endtable

$table
$stream
               add  reg3, r0, imm2
  LABEL2:mfc   reg4, state
               and  reg4, reg3
               beq  reg4, reg3, LABEL2
               nop
$format
               sleep  or  imm2
$endtable
```

【図２０】